いちばんわかりやすい

MS-DOS

マクロアセンブラ入門

実用プログラムを開発しながら
マシン語の知識が身につく本

藤田英時＝著



ナツメ社

[本書を読まれる方へ]

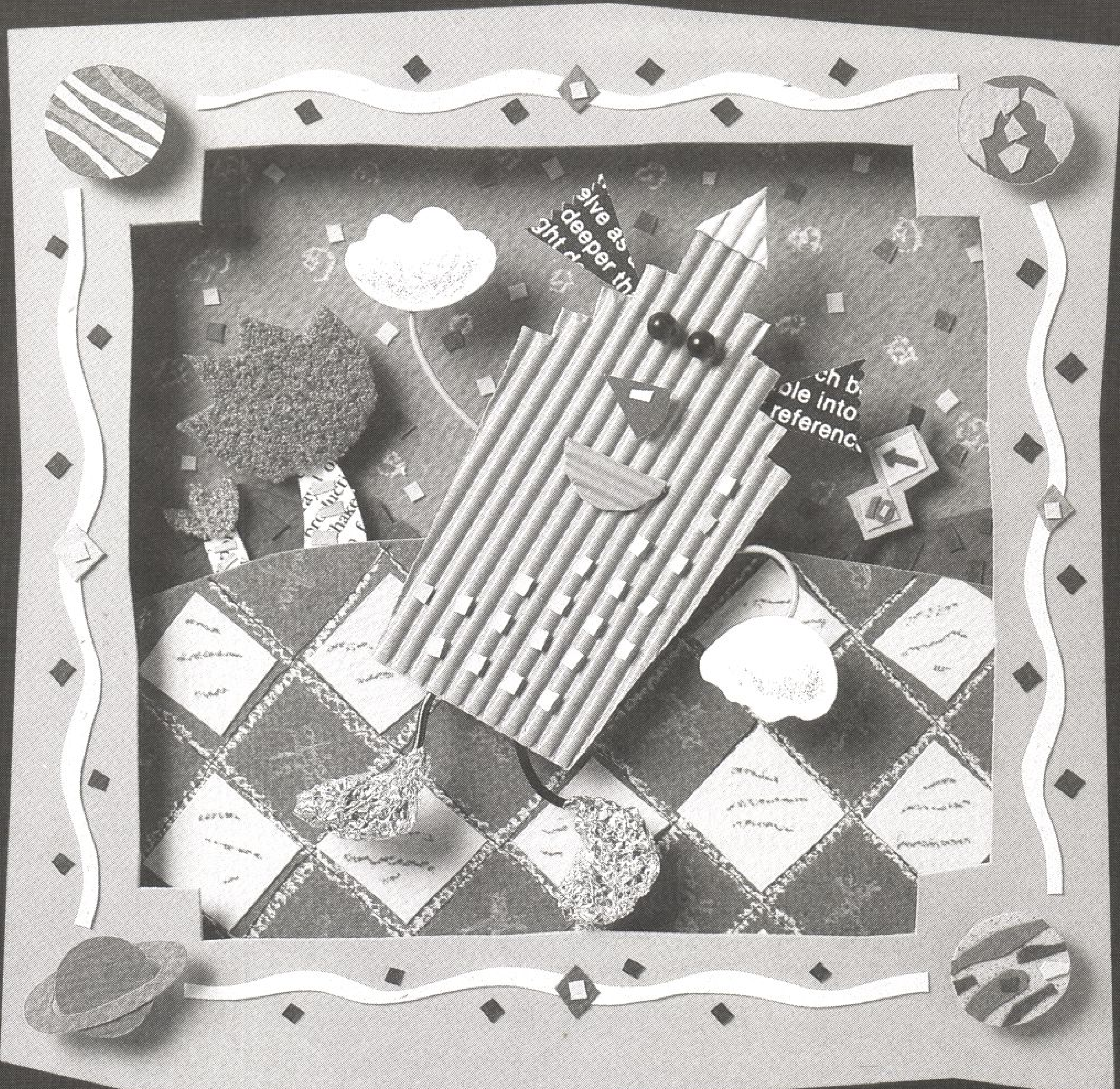
●あなたは「マシン語やアセンブリ言語はむずかしい」
という先入観を持っていませんか?
もし,そうであれば,この本の説明にそって,
実際にマシン語を"体験"してみることをおすすめします.
マシン語の世界に,自然に無理なく
入って行くことができるでしょう.
●本書では,まず体験[実行]してみて,概要をつかみ,
疑問点がでてきたところでそれらを解消し,
理解に導く,という解説をしています.
そして,マシン語などの基礎的な概念や機能,
使い方がわかったあとで,プログラムを開発していくという
アプローチをとっています.
この「体験,知識,理解から開発へ」が
本書のメインコンセプトですが,
次のような,類書には見られなかった解説方法も
とることで,よりわかりやすくしました.
▶実用的なプログラム例を全面に押し出して,
実用からマシン語の世界に入るようにした.
▶実用的なプログラムを作っていく過程のなかで,
アセンブリ言語によるプログラミングの技術が
自然に身につくように解説した.
▶アセンブリ言語のプログラムに対応する
BASIC言語のプログラムもあわせて紹介し,理解の一助とした.
▶マシン語の命令を個々に解説するのではなく,
あるひとかたまりの処理を行うなかで,
相互に関連づけて説明した.
▶初心者には不要と思われる命令や概念については,
大胆に切り捨て,混乱が起こらないようにした.

イラストレーション=千須和正之
カバー・デザイン=早瀬芳文

いちばんわかりやすい
MS-DOS マクロアセンブラ入門

実用プログラムを開発しながら
マシン語の知識が身につく本

藤田英時＝著



ナツメ社

●はじめに

▶なぜマシン語とアセンブリ言語なのか

パソコンのマシン語とアセンブリ言語を知ることにより、あなたはこれまでと違って、次のような幅広い知識と経験をもつようになります。

- 現在のコンピュータの原点・原理がわかる
- CPU の働きがわかる。
- パソコンが動く基本的なしくみがわかる。
- BIOS (基本入出力システム) がどんなものであるかわかる。
- リバースエンジニアリング (解析) の意味がわかる。
- OS の働きがわかる。
- 市販ソフトがどのように開発されているかわかる。
- BASIC 言語とのマシン語のかかわりがわかる。
- C 言語への理解がはやまる。
- プログラミングの原点がわかる。
- 簡単で実用的なマシン語プログラムが作れる。

▶本書の内容と特長

こうした点を考慮して本書では、世界中で最も普及している 16 ビット CPU(8086系) と OS(MS-DOS) を取り上げ、BASIC 言語で簡単なプログラミングができる方を対象に、アセンブラを使いマシン語プログラムを開発する方法をやさしくわかりやすく解説しています。

本書は、次のような方を想定して書いています。

- マシン語やアセンブリ言語などはまったくはじめての方。
- マシン語やアセンブリ言語をすこしかじった方。
- まえまえからマシン語プログラムが必要な方。
- 実用的なマシン語プログラムが必要な方。

内容は、大きく次の 3 部にわかれています。

第 1 部　マシン語とアセンブラを体験する
第 2 部　マシン語とアセンブラを理解する
第 3 部　マシン語をアセンブラで開発する

まず、あなたが実際に、マシン語とアセンブラを体験してみて、その概要をつかみ、疑問点や難解なところがでてきたところで、それらを解消し、理解に導く解説をしています。そしてマシン語やアセンブリ言語、アセンブラの基礎的な概念や機能、使い方がわかったあとで、プログラムをいろいろと開発していくというアプローチをとっています。

この「体験、知識、理解から開発へ」が本書のメインコンセプトですが、次のように他のマシン語入門書には見られなかった解説や説明の方法をとっています。

・実用的なプログラムを全面的に押し出して、実用からマシン語の世界に入るようにした。
・実用的なプログラムを作っていく過程で、アセンブリ言語による基礎的でしかも総合的なプログラミング技術が身につくように解説した。
・アセンブリ言語プログラムの解説は、それぞれ重要な部分をはじめに行い、最後にプログラム全体について行った。
・アセンブリ言語プログラムに対応するBASIC言語のプログラムもあわせて紹介し、説明を加えた。
・マシン語プログラム開発に必要なパソコンとOSの情報についてもあわせて紹介した。
・マシン語の個々の命令を解説せず、あるひとかたまりの処理を行うなかで、相互に関連づけて説明した。
・マシン語関連で最低限、知っておくべきことをまとめて解説した。
・操作の全体像やプログラムの実行結果を先に見せて、全体の内容がいちはやく把握できるようにした。
・図表、アセンブリ言語リストを豊富に掲載し、ポイントがひと目でわかるようにした。
・初心者には、不要と思われる命令や概念については、大胆に切り捨て、混乱が起こらないようにした。

本書の読み方としては、はじめから順次読み進んでいき、プログラムの入力や実行など、操作する場面に来たら実際にやってみてください。はやい理解が得られます。

▶必要なハードとソフト

本書を読んで実際に操作するにあたり、必要なハードとソフトは次のとおりです。

＜ハードウェア＞
- PC-9800シリーズのパソコン本体
- ディスプレイ
- プリンタ

＜ソフトウェア＞
- MS-DOSシステムディスク
  (バージョン2.11または3.10)
- MS-DOSマクロアセンブラ
- テキストエディタ（ワープロでも可）

本書により、マシン語の世界の面白さを味わい、幅広い知識と経験をもつことによって、実用的なプログラムを利用し、日々のパソコン操作が快適に、かつ効率よく運ぶよう願っています。

1987年8月
藤田　英時

アセンブリ言語
アセンブラ（コンパイラの事）
ソースファイル→マシン語ファイルに直す

いちばんわかりやすいMS-DOSマクロアセンブラ入門

目　次

第1部 マシン語とアセンブラを体験する

第2部 マシン語とアセンブラを理解する

第3部 マシン語をアセンブラで開発する

第2章　実用プログラムの開発

第3章　独自の実用コマンドを作る

第1部 マシン語とアセンブラを体験する

第1部では、マシン語の開発環境を整えて、とにかくマシン語とアセンブラを体験することにします。まずマシン語を実行し、次にアセンブラでマシン語を開発して、実行することで、おぼろげながら全体の概要をつかんでください。

第1部

第1章 開発環境を整備する

マシン語を開発するためには、まずその開発環境を整えておくことが大切です。この章では、自分専用の開発システムディスクを作ることについて考え、マシン語開発に必要なファイルについて知り、最後に用語を覚えます。

1・1 開発専用のシステムディスクを作る

MS-DOSのマシン語開発環境整備の第1歩として、自分専用のシステムディスクを作ることをお勧めします。オリジナルのMS-DOSシステムディスクには、普段あまり使わないコマンド・ファイルが入っています。そこで、よく使うコマンド・ファイルだけを抜き出し、それにマシン語開発に必要なソフトをあわせてシステムディスクにしたらどうでしょう。

■本書で必要なコマンド・ファイル

●コマンド・ファイルとは、拡張子にEXEまたはCOMが付いたもので実行できるマシン語ファイル。

よく使うコマンド・ファイルは、人によって違うと思いますが、次のようなものを選んでおけばよいでしょう（これらは、本書での操作に必要なコマンドです）。

■アセンブラ関連のファイル

●デバッガとは、マシン語プログラム(EXEやCOMファイル)を実行して、そのバグ(誤り)をみつけだすためのプログラム。

MASM. EXE（マシン語ファイル作成）
LINK. EXE（EXEファイル作成）
EXE2BIN. EXE（COMファイル作成）
SYMDEB. EXE（デバッガ, Ver. 3. 10の場合）
DEBUG. COM（デバッガ, Ver. 2. 11の場合）
EDLIN. EXE（テキスト・ファイル作成）

このうち、PC-9800シリーズ用のMS-DOSでは、MASM. EXEとLINK. EXEは別売になっています。

●スクリーン・エディタとは、画面上でカーソルを自由に動かしテキスト・ファイルを作成・編集するプログラム。

アセンブリ言語プログラムを作るときには、テキスト・ファイルが作れるソフトがぜひとも必要になります。EDLINでもかまいませんが、できたらスクリーン・エディタがあると便利です。また、テキスト・ファイルが作れるワープロでも結構です。

ちなみに、私はMS-DOSのシステムに日本語フロントプロセッサを組み込み、市販のスクリーン・エディタを利用しています。

■自分のシステム用ファイル

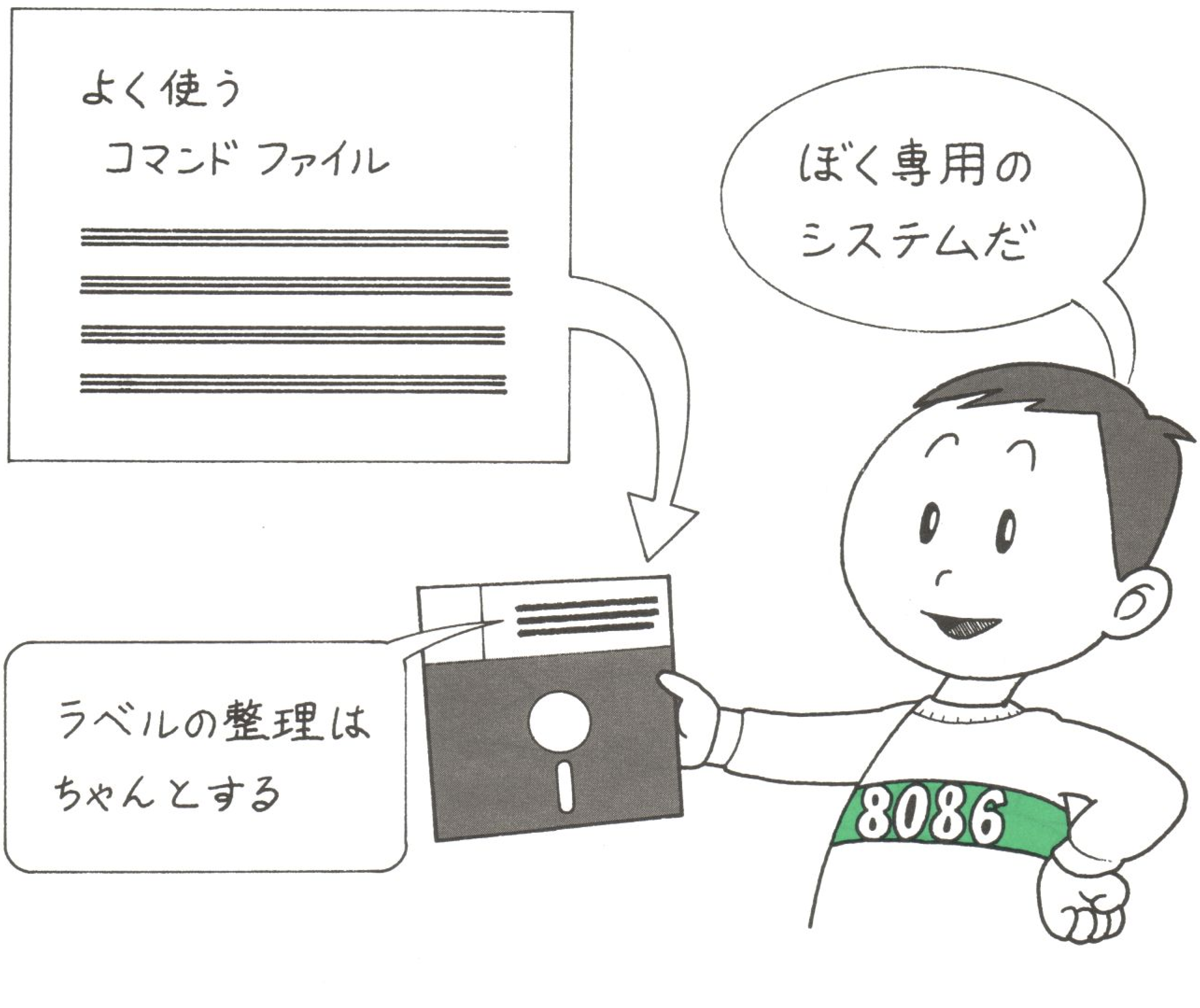
その他、自分のシステムに入れておきたいものとして、次のファイルがあります。

▶フロッピーディスク関連（バッチ・ファイルの例で使用）

FORMAT. EXE（フォーマット, Ver. 3. 10 の場合）
FORMAT. COM（フォーマット, Ver. 2. 11 の場合）
DISKCOPY. COM（バックアップ）

▶ファイル操作関連（バッチ・ファイルの例で使用）

FIND. EXE（検索）
MORE. COM（1 画面表示）
SORT. EXE（並べ換え）

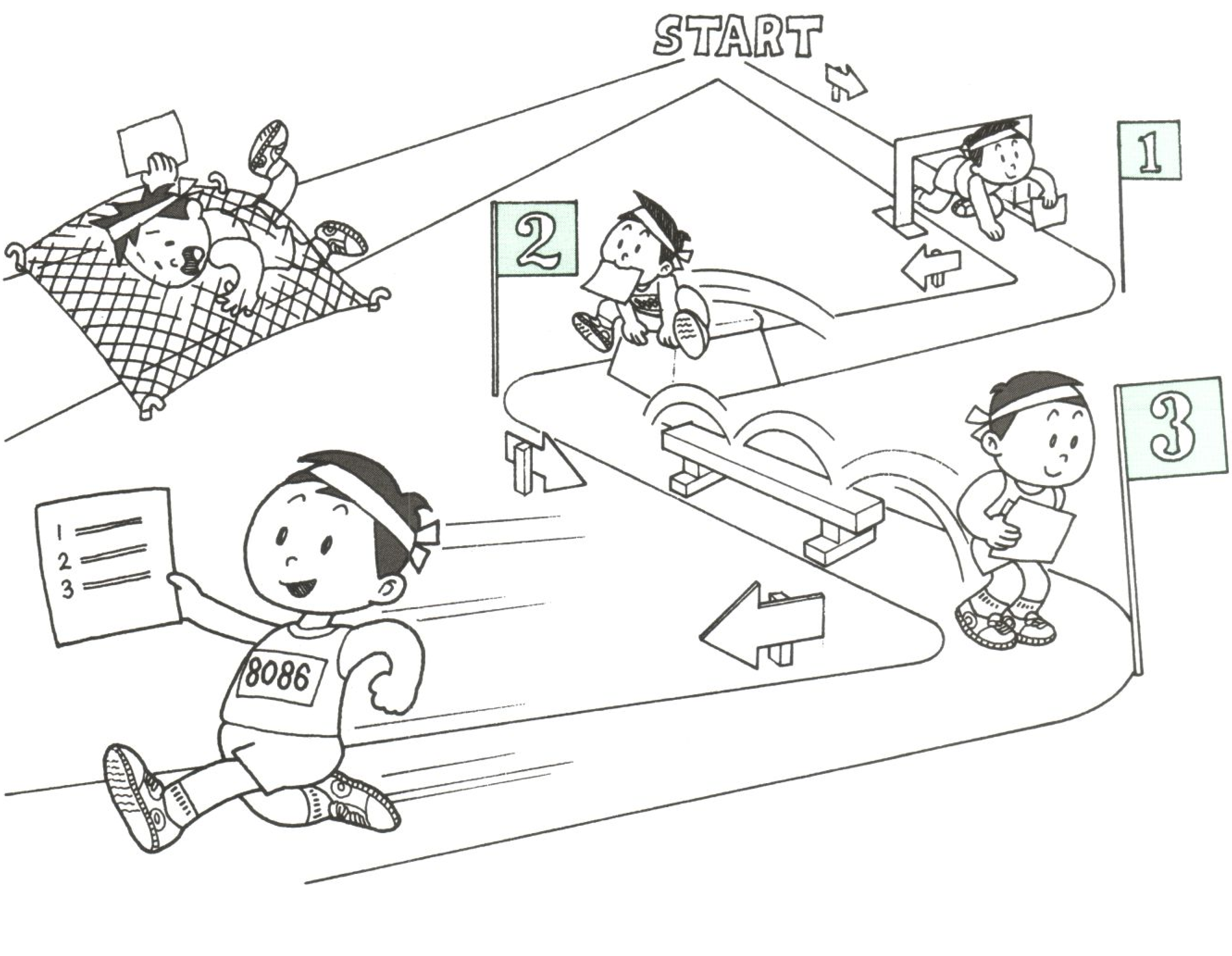


■アセンブラにいくつものプログラムが必要な理由

アセンブラでのプログラム開発には、前出のように、いくつものプログラムが必要です。それは、次のような手順を踏まなくてはならないからです。

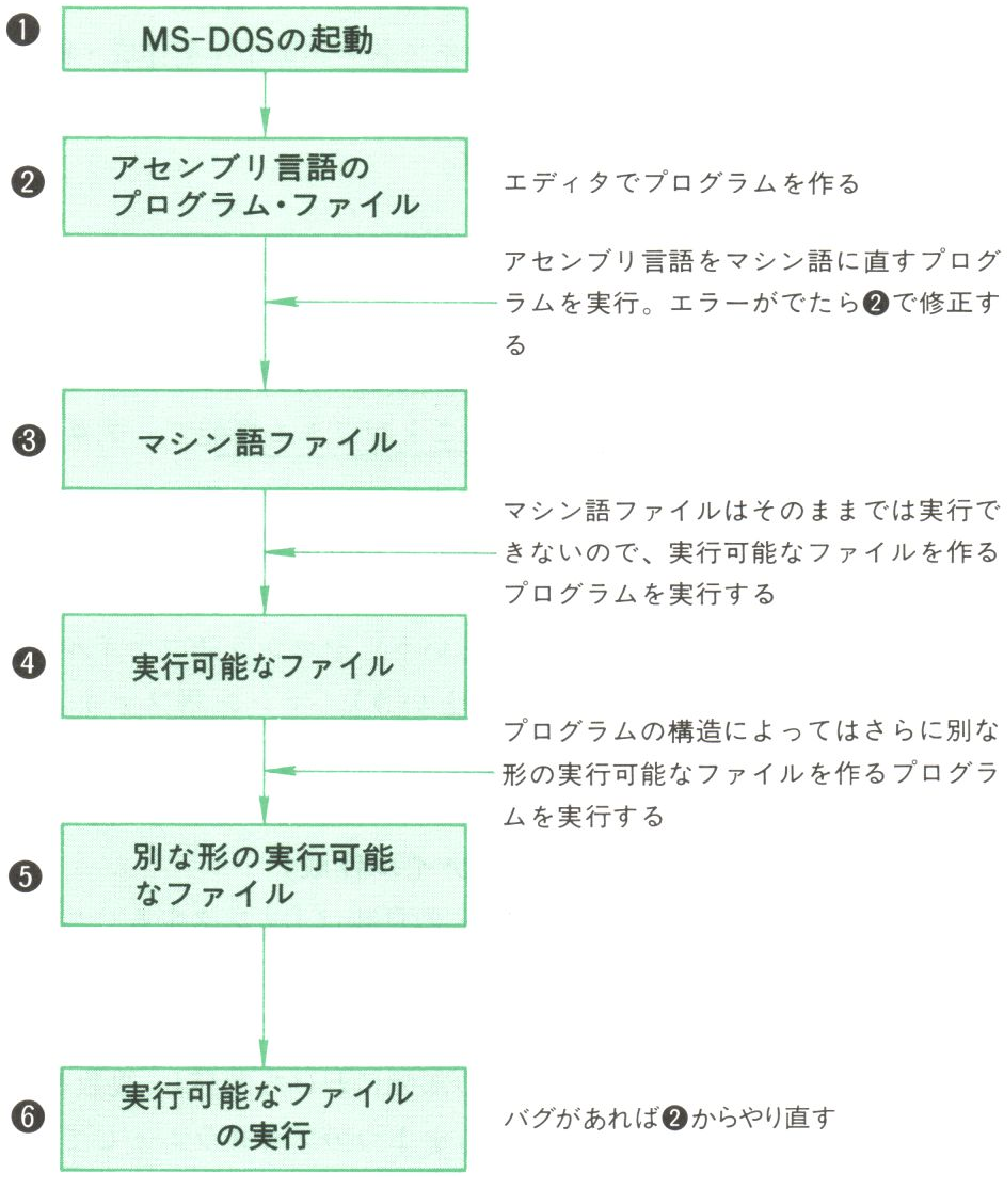
1. MS-DOS を起動する。
2. エディタでアセンブリ言語のプログラム・ファイルを作る。
3. プログラム・ファイルをマシン語ファイルに直す。
（ここでエラーが出たら、2 と 3 を繰り返す）
4. マシン語ファイルを実行可能なファイルに直す。
5. 開発するマシン語プログラムの構造によっては、さらに実行可能なファイルを別な形の実行可能なファイルに直す。
6. 実行可能なファイルを実行する。期待どおりの結果が得られない場合は、バグをみつけるプログラムを実行して調べる。
7. バグがみつかれば、2 からの手順を繰り返す。

●実行可能なファイルには、EXE ファイルと COM ファイルとがある。



このステップを図で表わすと、次のようになります。

■マシン語開発・実行までの手順



これはとても複雑と感じるかも知れませんが、これがプログラム開発の本来の手順なのです。BASIC 言語でプログラムを組んで、インタプリタで実行するのに慣れている方にとっては、たしかに面倒でしょう。

しかし、BASIC 言語でも、コンパイラを使うと図のような手順を踏まなくてはなりません。C 言語や COBOL でのプログラミングも同じです。

1・2 アセンブラ関連ファイルの知識

ここで、さっき挙げた必要なファイルの説明を簡単にしておきます。

▶エディタ（アセンブリ言語ファイル作成）

EDITORのことで、テキスト・ファイルを作成・編集するソフト。プログラムを書くためのもので、スクリーン・エディタで使い勝手がよいものが開発に最適です。

▶ MASM. EXE（マシン語ファイル作成）

Macro ASseMblerのことで、マクロ機能をもったアセンブラです。マクロ機能とは、あるまとまった処理をするいくつかの命令を、1つの命令として取り扱うことができる機能で、プログラム作成の効率が上がるものです。

アセンブラとは、アセンブリ言語で書いたプログラム・ファイル(拡張子は、ASMでなければならない。これをもとのプログラムということでソース・プログラムという）をマシン語ファイルに直すものです（これをアセンブルするという)。マシン語ファイルの拡張子にはOBJect（マシン語）を表わすOBJが自動的に付けられます。

●アセンブラは、アセンブリ言語で書いたプログラム・ファイルをマシン語ファイルに直す。

●マクロアセンブラでは、ソース・プログラムには必ずASMという拡張子を付ける。アセンブルされたらOBJファイルができる。

▶ LINK. EXE（EXEファイル作成）

LINKerのことで、もとの意味は「リンクする、つなぎあわせる」ということです。MASMでマシン語になったファイル（OBJファイル）を実行可能なEXEファイルにします。

プログラムをいくつかの部分にわけて開発し、複数のOBJファイルができた場合には、これらを1つのプログラムとして「つなぎあわせ」ることもできます。じつは、これがLINKの本来の働きといえます。もちろん、OBJファイルが1つのときでも、必ずLINKを使ってEXEファイルにしなければなりません。

●LINKは、OBJファイルをEXEファイルに直す。

OBJ
↓
EXE

▶ EXE2BIN. EXE（COMファイル作成）

これは、EXEファイルをCOMファイルに変換します。MS-DOSで実行可能なマシン語ファイルは、拡張子にCOMかEXEが付いたものです。それぞれ、次のような意味です。

●EXE2BINは、EXEファイルをCOMファイルに直す。

EXE
↓
COM

COM ：COMmand File のことで、直訳すると命令ファイルで、ファイル名を入力すると、それが実行されるものです。

EXE ：EXEcutable File のことで、直訳すると実行可能ファイルで、これもファイル名を入力すると、それが実行されます。

両者はプログラムの実行という点では、なんら変わりがありませんが、違いは、プログラムの構造です。一般にCOMファイルの方が、プログラム・サイズが小さく、EXEの方が大きいものです。また、COMファイルの方が簡単にプログラムを作れます。本書では、入門なのですべてCOMファイルの作り方を説明します。

▶ SYMDEB. EXE（デバッガ, Ver. 3.10の場合）

SYMbolic DEBugerのことで、シンボルすなわちラベル（名前）が使えて、デバッグをするソフト（デバッガ）です。COMファイルやEXEファイルを実行しながら、その過程を追っていけ、誤りを見つけることができます。

●ラベルとは、飛び先やメモリの特定位置に付ける名前のこと。

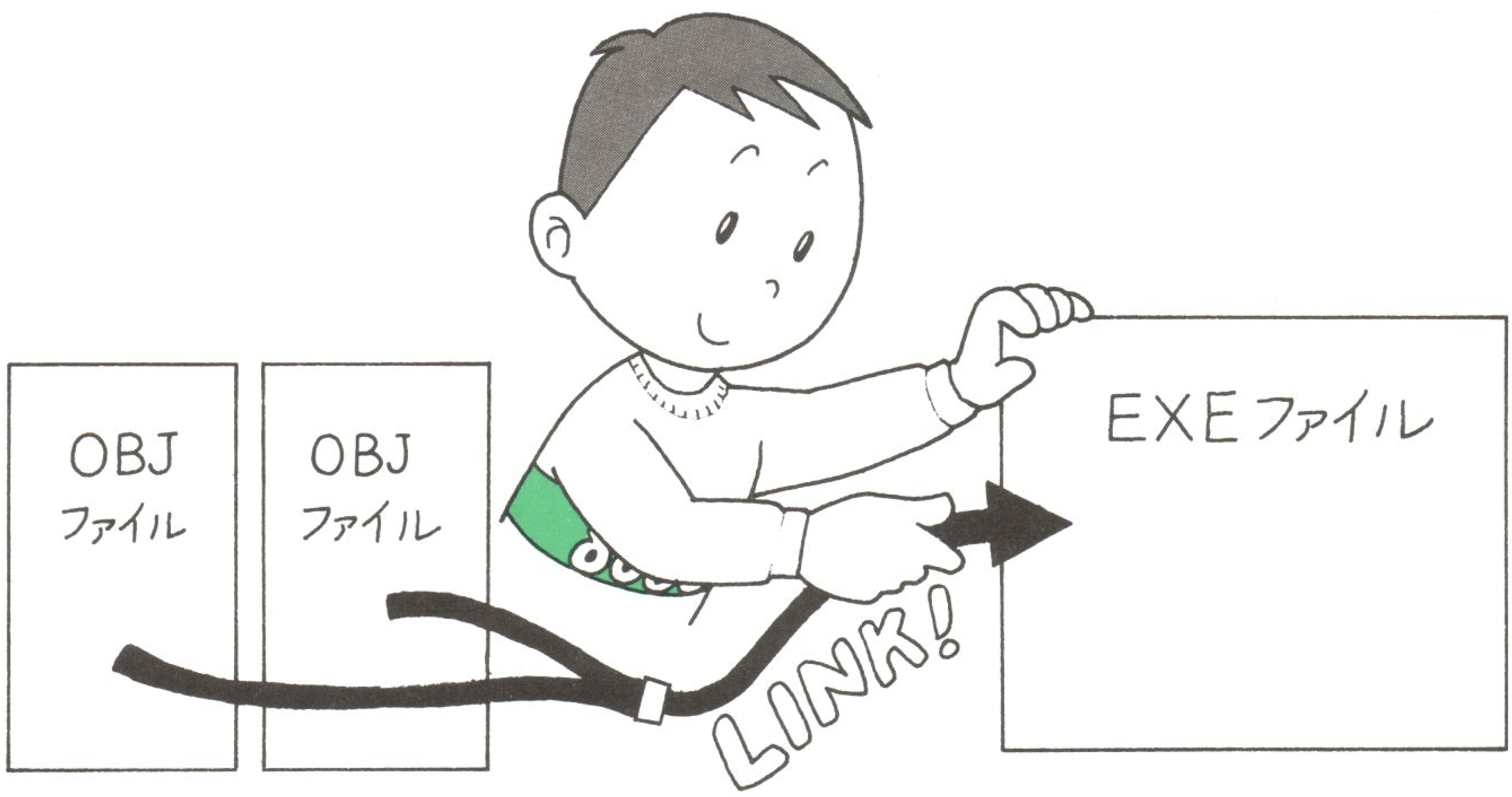
また、アセンブルとアンアセンブル（Unassemble：マシン語をアセンブリ言語に直す）の機能をもっています。

●アンアセンブルは、アセンブルと逆の動作をする。マシン語をアセンブリ言語に直す。

▶ DEBUG. COM（デバッガ, Ver. 2.11の場合）

MS-DOS Ver. 2. 11のデバッガです。これは1つ前のバージョンですので、機能はSYMDEBとほとんど同じですが、ラベルを使ってのデバッグはできません。

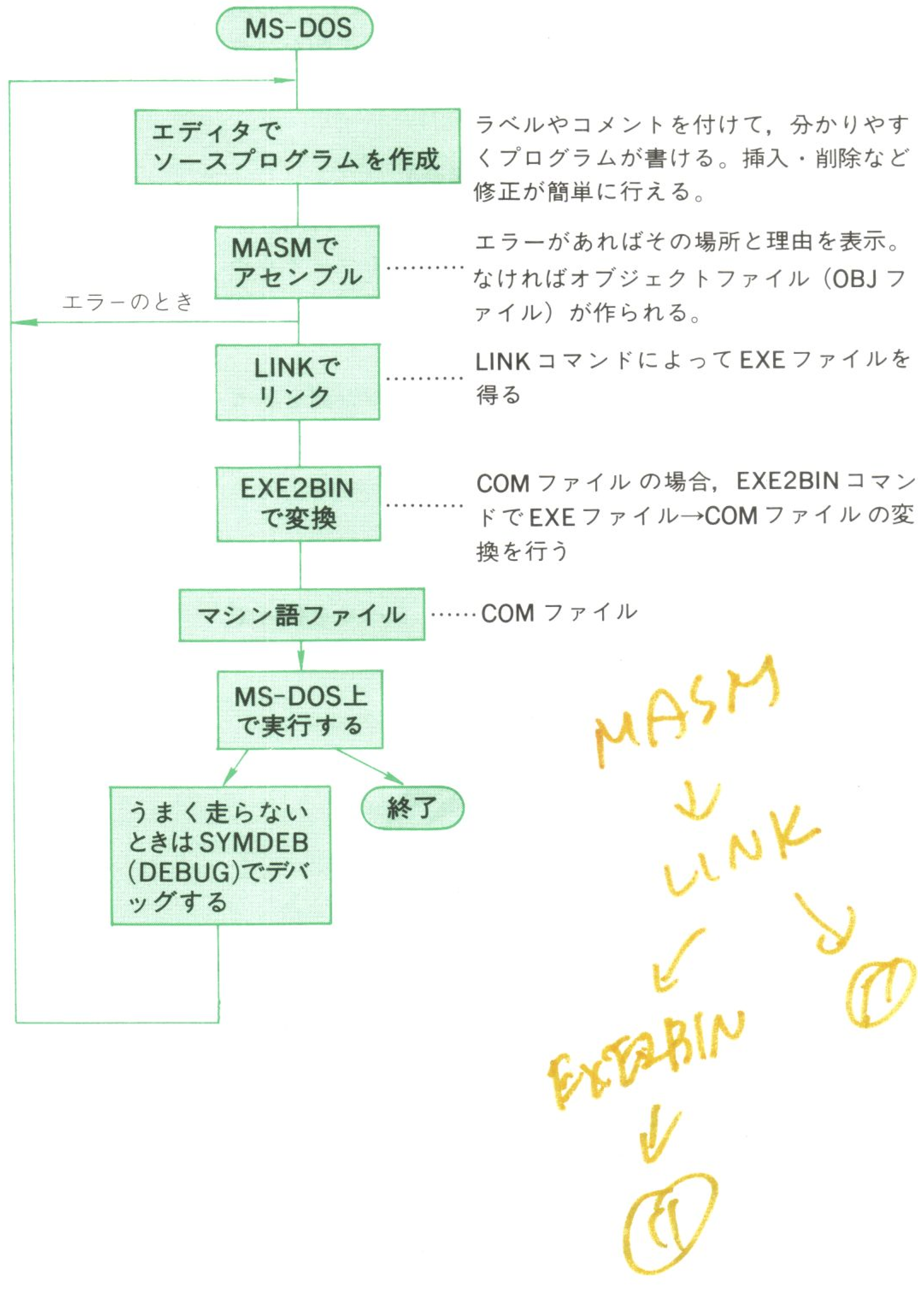
●SYMDEB では起動メッセージが表示されるがDEBUGでは、これが表示されない。



なお、本書ではSYMDEBを使って説明をしますが、操作はDEBUGと同じですので、DEBUGをお持ちの方は、起動のときだけSYMDEBをDEBUGとしてください。

これらのファイルとプログラム開発の関連を図で表わすと、次のようになります。

■各ファイルとプログラム開発との関連



1・3 マシン語・アセンブラ関連用語のまとめ

本格的な用語の解説は、該当個所で行いますが、ここでとりあえず、これから読み進むのに知っておくと理解の助けとなる用語を、本章のまとめとして簡単に説明しておきます。

▶マシン語

CPU が実行できる唯一の言語。CPU によって異なる。

▶アセンブリ言語

マシン語の命令と 1 対 1 に対応した言語で、数字だけのマシン語と違って、人間にも意味がつかめるため比較的わかりやすい。CPU によって異なる。

▶アセンブラ

アセンブリ言語をマシン語に直すプログラム。

▶アセンブル

アセンブリ言語をマシン語に直す。

▶アンアセンブル

マシン語をアセンブリ言語に直す。

▶インタプリタ

BASIC などの高級言語をその都度解釈・実行するプログラム。

▶コンパイラ

BASIC などの高級言語をマシン語に直すプログラム。この場合、高級言語の 1 命令とマシン語の 1 命令とは対応しない。

▶マシン語ファイル

そのファイルを入力すると実行可能なファイル、ただしオブジェクト・ファイルはマシン語だが実行できない。MS-DOS では、EXE ファイルと COM ファイルがある。

comファイルに変換する
TEST.EXE
↓
EXE2BIN
↓
TEST.COM

▶ソース・ファイル（ソース・プログラム）

アセンブルの対象となるもとのファイル（プログラム）

▶エディタ

ソース・ファイル（プログラム）を作るプログラム。

▶オブジェクト・ファイル

アセンブルされてマシン語になったファイル。ただし、リンクしてEXEファイルにしないと実行できない。

▶リンク

マシン語になったファイルをつなぎあわせ、実行可能なファイル(EXEファイル）にする。

▶デバッガ

マシン語プログラムのバグを取り除くプログラム。

第1部

第2章 マシン語をはじめて実行する

この章では、デバッガ（SYMDEB, DEBUG）を使ってアセンブリ言語でプログラムを入力し、それをマシン語にして、マシン語を実行してみることにします。論よりもとにかくマシン語を体験することが、その理解への第1歩だからです。そして体験したことについて簡単な説明をします。

2・1 プログラムの内容

まず、なにはともあれマシン語がどういうものか実行してみましょう。ここでは、MS-DOSが起動したときに、ファンクションキーに設定されている内容を読み取るマシン語プログラムを作って実行してみます。プログラムが短く、実行結果がひと目でわかるので、これを選びました。

結果としては、次のようにファンクションキーの内容が読み取れます。

●キーの内容は、通常表示されているものと少し違います。よけいな記号や文字は無視してください。

■読み取ったファンクションキーの内容

```
~ C1   .S........
~ CU   .T........
~ CA   .U........
~ S1   .V........
~ SU   .W........
~VOID  .E........
~NWL   .J........
~INS   .P........
~REP   .Q........
~ ^Z   .Z........
dir a:..........
dir b:..........
copy ..........
del ...........
ren ...........
chkdsk a:......
chkdsk b:......
type ..........
date...........
time...........
```

(上10行：f･1〜f･10、下10行：SHIFT＋f･1〜f･10)

■実際のファンクションキーの内容(初期設定)

C1	CU	CA	S1	SU	VOID	NWL	INS	REP	^Z
*dir a:	dir b:	copy	del	ren	chkdsk	chkdsk	type	date	time

2・2 実行の手順

プログラムの入力から実行、そして結果を見るまでの手順は、次のとおりです。

1. MS-DOS を起動する。
2. デバッガ（SYMDEB または DEBUG）を起動する。
3. アセンブル・コマンドを実行する。
4. アセンブリ言語を入力する。
   （1 行入力するたびに自動的にマシン語に直される）
5. アンアセンブル・コマンドを実行する。
   （入力したアセンブリ言語とそれに対するマシン語を表示）
6. ファンクションキーの内容を読み取るメモリの内容を表示する
   （なにも入ってないことを確認する）。
7. マシン語を実行する。
8. 再び、メモリの内容を表示する。
   （ファンクションキーの内容が入っている）
9. デバッガを終了する。

2・3 実行の実際

MS-DOS のコマンド待ちの状態で、

A＞SYMDEB（または DEBUG）

とするとデバッガが起動します。

－（ハイフン）は、デバッガのプロンプトです。これが表示されたら、デバッガのコマンドが入力できます。まず、アセンブルするため、そのコマンド A（Assemble）を入力します。

●アドレスとは、メモリの特定位置を示すために付けられた番号。

すると左側にマシン語を格納するアドレス（メモリの位置を表わすための番号）が自動的に表示されます。このアドレスは、MS-DOS を起動した環境や 98 本体の実装メモリサイズによって異なります。

次のアセンブリ言語を 1 行ずつ、入力します。

●これらが 8086 のアセンブリ言語です。

```
MOV   DX, 200
MOV   CL, C
MOV   AX, 0
INT   DC
```

最後は、ただリターンキーだけを押します。

1 行入力するたびに、アドレスが増えていきます。これは、入力したアセンブリ言語がマシン語になってメモリに格納されていくからです。入力ミスをすれば、アセンブルされませんので、BS キーで修正するか、もう 1 度入力しなおします。

次の画面で、　　　　部を入力します。

■デバッガの起動とアセンブル

```
A>SYMDEB ⏎ ── デバッガの起動
Microsoft Symbolic Debug Utility        ┐ シンボリック
Version 3.01                            │ デバッガの
(C)Copyright Microsoft Corp 1984, 1985  │ 起動メッセージ
Processor is [8086]                     ┘
-A ⏎ ── アセンブル
1A10:0100 MOV DX,200 ⏎  ┐
1A10:0103 MOV CL,C ⏎    │ アセンブリ言語を入力
1A10:0105 MOV AX,0 ⏎    │
1A10:0108 INT DC ⏎      ┘
1A10:010A ⏎ ── 入力終了
```

デバッガのプロンプト

アドレス表示

● DEBUG では起動メッセージが表示されない。

●アセンブルするときは、**アドレスが 100 からはじまる**ようになっています。アドレスを指定して、

A100

としても同じです。

これで、マシン語がメモリに格納されました。確認のために次のようにしてください。

-U100, 109

●アンアセンブル・コマンドは、アンアセンブルする開始アドレスと終了アドレスを指定する。ここでは、100 から 109 まで。

U は、Unassemble (アンアセンブル) コマンドで、メモリの中のマシン語をアセンブリ言語に直して表示します。

■アンアセンブル

```
-U100,109 ⏎ ── アンアセンブル
1A10:0100 BA0002 ──── MOV DX,0200
1A10:0103 B10C   ──── MOV CL,0C
1A10:0105 B80000 ──── MOV AX,0000
1A10:0108 CDDC   ──── INT DC
-
```

アドレス表示　マシン語　アセンブリ言語

ご覧のように、マシン語とアセンブリ言語が 1 行ずつ対応して表示されています。

実行の前に、ファンクションキーの内容を入れるメモリの内容を見ておきましょう。そのアドレスは、16 進数で 200 から 33F までです。

次のようにしてください。

● 16 進数は、数字と A から F までを使って表現する数の数え方で数が 16 個集まると 1 桁くりあがる。

●ディスプレイ・コマンドも表示する開始アドレスと終了アドレスを指定する。ここでは、200 から 33F まで。

-D200, 33F

D は、Display（表示）コマンドで、メモリの内容を 16 進数とアスキー文字で表示します。

■メモリ内容表示

```
-D200,33F ⏎
1A10:0200  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0280  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0290  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:02A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:02B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:02C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:02D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:02E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:02F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0300  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0310  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0320  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1A10:0330  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

メモリ内容表示

アドレス表示

メモリ内容（16進表示）

メモリ内容（アスキー表示）

●実行コマンドは、通常実行を終了するアドレスを指定します。ここでは 10A。実行開始アドレスは、はじめて実行するときは 100 になっています。なお、実行開始アドレスと終了アドレスを指定するには、次のようにします。

G=100,10A

メモリになにも入っていないときは、16 進数では 00、アスキー文字では・（点）で表示されます。今、なにも入っていません。ここに、ファンクションキーの内容を入れようというわけです。

さて、実行です。次のようにします。

-G10A

G は、GO（実行）コマンドで、アドレスこの場合 100 から実行開始し、10A で実行を止めます。

■マシン語の実行

マシン語の実行

8086がもつ変数の内容一覧

```
-G10A
AX=0000  BX=0000  CX=000C  DX=0200  SP=E4DF  BP=0000  SI=0000  DI=0000
DS=1A10  ES=1A10  SS=1A10  CS=1A10  IP=010A   NV UP EI PL NZ NA PO NC
1A10:010A 0000          ADD     [BX+SI],AL                        DS;00
```

実行を止めたアドレス

実行を止めたところの内容

実行が終わりました。一瞬のことです。実行のあと、なにか表示されていますが、ここでは無視してください。

結果を見てみましょう。次のようにしてください。

−D200, 33F

■実行結果の表示

メモリ内容表示

```
-D200,33F
1A10:0200  FE 20 43 31 20 20 1B 53-00 00 00 00 00 00 00 00  ~ C1  .S.......
1A10:0210  FE 20 43 55 20 20 1B 54-00 00 00 00 00 00 00 00  ~ CU  .T.......
1A10:0220  FE 20 43 41 20 20 1B 55-00 00 00 00 00 00 00 00  ~ CA  .U.......
1A10:0230  FE 20 53 31 20 20 1B 56-00 00 00 00 00 00 00 00  ~ S1  .V.......
1A10:0240  FE 20 53 55 20 20 1B 57-00 00 00 00 00 00 00 00  ~ SU  .W.......
1A10:0250  FE 56 4F 49 44 20 1B 45-00 00 00 00 00 00 00 00  ~VOID .E.......
1A10:0260  FE 4E 57 4C 20 20 1B 4A-00 00 00 00 00 00 00 00  ~NWL  .J.......
1A10:0270  FE 49 4E 53 20 20 1B 50-00 00 00 00 00 00 00 00  ~INS  .P.......
1A10:0280  FE 52 45 50 20 20 1B 51-00 00 00 00 00 00 00 00  ~REP  .Q.......
1A10:0290  FE 20 5E 5A 20 20 1B 5A-00 00 00 00 00 00 00 00  ~ ^Z  .Z.......
1A10:02A0  64 69 72 20 61 3A 0D 00-00 00 00 00 00 00 00 00  dir a:..........
1A10:02B0  64 69 72 20 62 3A 0D 00-00 00 00 00 00 00 00 00  dir b:..........
1A10:02C0  63 6F 70 79 20 00 00 00-00 00 00 00 00 00 00 00  copy ...........
1A10:02D0  64 65 6C 20 00 00 00 00-00 00 00 00 00 00 00 00  del ............
1A10:02E0  72 65 6E 20 00 00 00 00-00 00 00 00 00 00 00 00  ren ............
1A10:02F0  63 68 6B 64 73 6B 20 61-3A 0D 00 00 00 00 00 00  chkdsk a:.......
1A10:0300  63 68 6B 64 73 6B 20 62-3A 0D 00 00 00 00 00 00  chkdsk b:.......
1A10:0310  74 79 70 65 20 00 00 00-00 00 00 00 00 00 00 00  type ...........
1A10:0320  64 61 74 65 0D 00 00 00-00 00 00 00 00 00 00 00  date............
1A10:0330  74 69 6D 65 0D 00 00 00-00 00 00 00 00 00 00 00  time............
```

メモリ内容(16進表示)

メモリ内容
(アスキー表示)

英数字のら列で、なんのことかわかりませんが、右端を見てください。見慣れた文字があります。そうです、MS-DOSで通常設定されているファンクションキーの内容です。はじめの10個の内容とSHIFTを押して使うときの内容10個があります。

これで、読者は8086のマシン語を実行したことになるのです。ファンクションキーの内容がうまく表示されない場合はシステムをリセットして、もう1度はじめからやり直してください。

最後に、

−Q

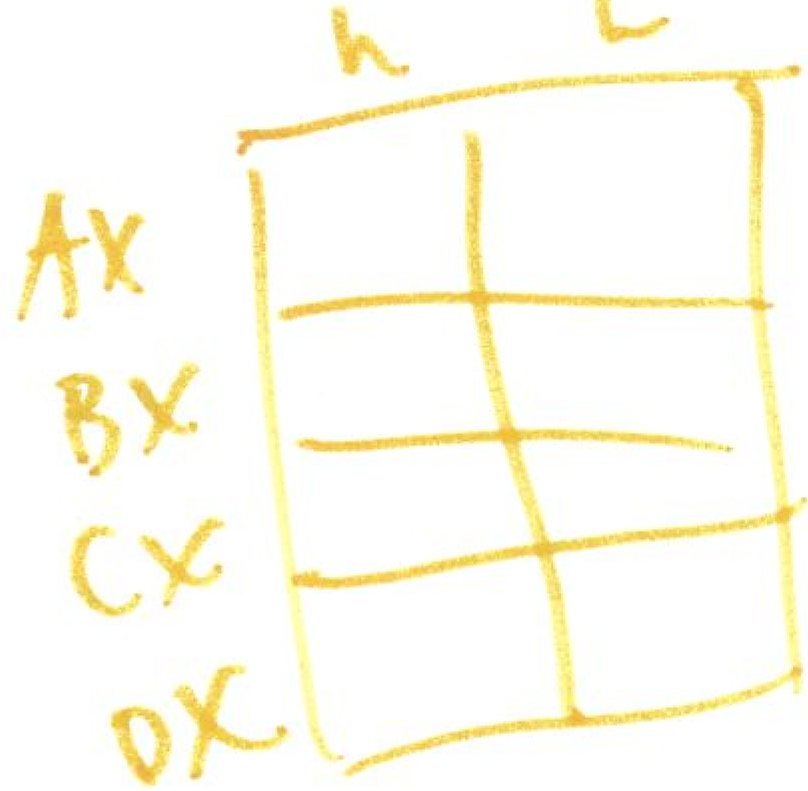
とすると、A>が表示されて、コマンド待ちに戻ります。Qは、Quit（止める）でデバッガの終了コマンドです。

■デバッガの終了

```
-Q ⏎ —— デバッガの終了
A>
```

2・4 操作内容の説明

では、簡単にこれまで行ってきた操作のひとつひとつについて、説明しますが、この時点では、よくわからなくても結構です。マシン語を実行することの概要をつかめば大丈夫です。



▶ SYMDEB

デバッガの起動 (SYMbolic　DEBuger)

デバッガの起動メッセージが表示される。

▶ A

アセンブル (Assemble) するデバッガの命令。

これでアセンブリ言語をマシン語に直す。

▶ 1A10:100

アセンブルされたマシン語が格納されるメモリの先頭アドレス。アドレスとは、メモリに順番に番号を付けたもの。このアドレスは16進数で表わされ、:(コロン)で区切られた2つの数が表示されているが、8086では、この2つのアドレスの組合わせにより、実際のアドレスを表わす。なお、: (コロン) の左側の数字はユーザーメモリの実装量やシステムの起動状態によって異なる。

●アドレスの見方は、「第2部1−4　セグメントの概念」(72ページ)参照。

▶ MOV　DX, 200

DXは、8086が持つ変数のようなもの。ここに16進数で200を入れるという8086のアセンブリ言語の命令。これは、 ファンクションキーの内容をメモリ・アドレス200から格納する設定をしている。MOVとは、MOVe (移す) のこと。

●8086のもつ変数のようなもの(これをレジスタという)については、「第2部1−4　レジスタとは」(69ページ)と「1−5　レジスタの機能と役割」(80ページ) を参照。

▶ MOV　CL, C

CLも8086が持つ変数のようなもの。ここに16進数でC (10進数で12) を入れる、8086のアセンブリ言語の命令。

▶ MOV　AX, 0

8086が持つ変数のようなものAXに、0を入れる。8086のアセンブリ言語の命令。

● INT とはプログラムの中の命令で実行できる割り込み処理で、「第２部 1−4　ソフトウェア割り込み」（74ページ）参照。

▶ INT DC

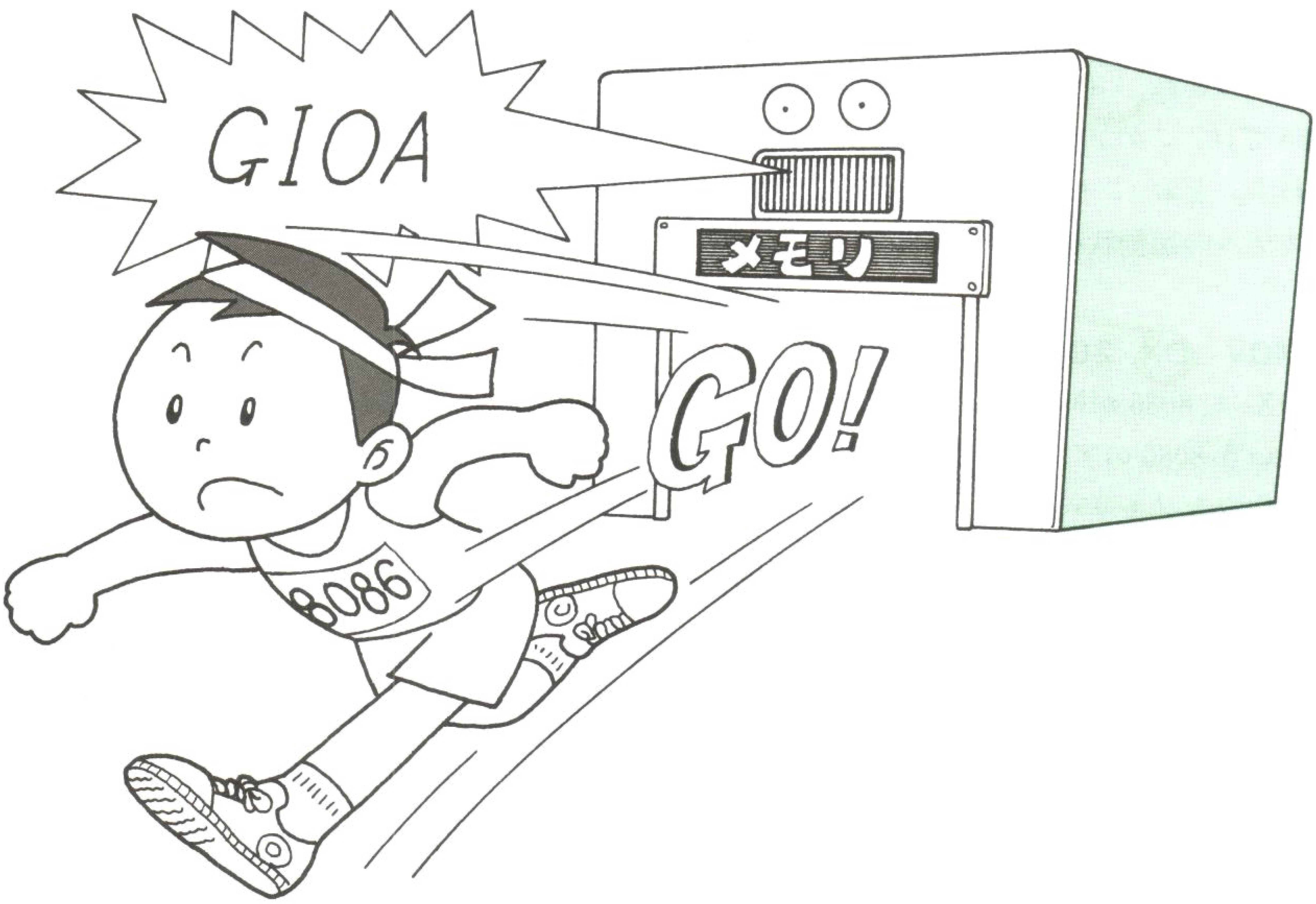
MS−DOS の持つファンクションキー読み書き機能を実行する。8086 のアセンブリ言語の命令。INT（イント）とは、INTerrupt（割り込み）で、割り込みで実行される命令。

▶ U100, 10A

アンアセンブル（Unassemble）するデバッガの命令。16 進数でアドレス 100 から 10A までをアンアセンブルする。

▶ G10A

メモリに格納されているマシン語を実行するデバッガの命令。Go（行く）ということで、実行する意味。アドレス 10A（16 進数）で実行を停止する。



▶ AX＝0000　BX＝0000　CX＝000C など

8086 が持つ変数のようなもの（レジスタという）の値が表示されている。実行結果がこれでわかる。

▶ D200，33F

メモリに格納されている内容を表示するデバッガの命令。Display（表示する）ということで、アドレス 200（16 進数）から 33F（16 進数）までを表示する。

▶ Q

デバッガを終了する命令。Quit（止める）ということ。

2・5 デバッガからアセンブラへ

ここでは、デバッガを使ってアセンブリ言語を入力し、マシン語を実行しましたが、デバッガの本来の機能は、マシン語ファイルになったもの（EXE や COM ファイル）の実行を追っていきながら、バグをみつけることにあります。

また、デバッガでは、簡単なアセンブリ言語ならば入力して、実行することができますが、少し長いプログラムや修正が必要なプログラムでは、もうお手上げです。それは、1行ずつの入力しかできず、行の挿入・削除ができないためです。

そこで、テキスト・エディタを使って、アセンブリ言語でプログラムを書き、アセンブラでマシン語に直すということになるわけです。次章では、これにチャレンジします。

第1部

第3章 アセンブラをはじめて使う

ここでも、細かいことは抜きにして、とにかくマクロアセンブラを使ってプログラムを作ってみましょう。エディタでのソースプログラム作成から、アセンブルして、マシン語ファイルの実行までを実用的な例で解説します。

3・1 プログラムの内容

●バッチ・ファイルとは、コマンドを１つ以上連ねて書いたファイルで、そのファイル名を入力すると、書いたコマンドを一括して実行できる。拡張子には、BAT を付ける。

MS-DOS では、バッチ処理が行えて便利ですが、バッチ・ファイルの中で処理の選択が思うようにできません。そこで、YES／NO の２者選択を行う機能をバッチ・ファイルに付加できたらいいと思いませんか。

この機能をマシン語で作ろうというわけです。実用的なマシン語プログラムを作ると、早くその作り方が身につくものです。

そのバッチ・ファイルの実行例をまず、紹介しましょう。

・拡張子を指定してドライブＢにあるファイルを表示する。

■拡張子を指定してファイルを表示

```
A>FDISP BAT ⏎

A>ECHO OFF
********** ファイル表示 **********
ドライブＢに表示したいディスクを入れてください.
準備ができたらどれかキーを押してください.
-----------------------------------------------
FDISP     BAT      413  87-07-14    1:10
FILES     BAT      657  87-07-09    1:59
FTYPE     BAT      427  87-07-13   23:26
処理を続けますか（Ｙ／Ｎ）？
Y-----------------------------------------------
ドライブＢに表示したいディスクを入れてください.
準備ができたらどれかキーを押してください.
-----------------------------------------------
123SYS    BAT       32  87-07-06   15:44
DBASESYS  BAT       33  87-07-06   15:44
CONSET    BAT      559  87-07-02   22:42
JXWSYS    BAT       30  87-07-06   15:43
処理を続けますか（Ｙ／Ｎ）？
N
A>
```

拡張子BAT（バッチ・ファイル）を表示する

YES/NO選択でYを入力

YES/NO選択でNを入力

「処理を続けますか（Y／N）?」のところで、ドライブＢのディスクを差し替え、「Y」を押して、処理を続けています。

・日付けを指定してファイルを表示する。

■日付けを指定してファイルを表示

```
A>FDISP 7-22 ⏎

A>ECHO OFF
********** ファイル表示 **********
ドライブBに表示したいディスクを入れてください.
準備ができたらどれかキーを押してください.
----------------------------------------------
YESNO    ASM      968  87-07-22  16:42
YESNO    OBJ       76  87-07-22  16:45
処理を続けますか（Y／N）？
Y
----------------------------------------------
ドライブBに表示したいディスクを入れてください.
準備ができたらどれかキーを押してください.
----------------------------------------------
MACHINE          7039  87-07-22  16:11
MACHINE  1       7919  87-07-22  16:31
処理を続けますか（Y／N）？
N
A>
```

7月22日のファイルを表示する

YES/NO選択でYを入力

YES/NO選択でNを入力

「処理を続けますか（Y／N）?」のところで、ドライブBのディスクを差し替え、「Y」を押して、処理を続けています。

このバッチ処理内容は、ドライブBにあるファイルをいろいろな条件を付けて表示するもので、ディスクを差し替えて、処理が続けられます。バッチ・ファイルの内容は、次頁のとおりです（ファイル名は、FDISP. BAT：File　DISPlay の略）。

■YES/NO選択ができるバッチ・ファイル

```
ECHO OFF
CLS
ECHO ********** ファイル表示 **********
IF "%1"=="" GOTO END
:START
  ECHO ドライブBに表示したいディスクを入れてください.
  PAUSE
  ECHO -----------------------------------------------
  DIR B: | FIND "%1"
REM --- YES／NO判断 ---
   ECHO 処理を続けますか（Y／N）？
   YESNO ―――――――――― YESNO.COMというマシン語ファイル
   IF ERRORLEVEL 1 GOTO END
   ECHO -----------------------------------------------
   GOTO START
:END
```

この中のYESNOがマシン語ファイル（YESNO.COM）で、キーボードから入力されたキーの内容をバッチ・ファイルに渡す役割を果たします。

IF ERRORLEVELとマシン語との関係

このバッチ・ファイルでは、IF ERRORLEVELを使って、キーボードから入力されたキーの内容がわかるようになっています。これは、もともとはMS-DOSのコマンド処理でエラーが生じたときのエラーの内容（異常終了か正常終了かなど）を判断する機能です。

●IF ERRORLEVELは、エラーコードを判定するバッチ・ファイル用のコマンドで、条件により処理を分岐させることができる。Yキーが入力されたら0,Nキーが押されたら1というようにエラーコードを設定して、バッチ・ファイルに戻し、これをIF ERRORLEVELで判断することができる。

ここでは、Y／Nのキー入力に応じてコード（0か1）を出すプログラムを作り、バッチ処理の中で、IF ERRORLEVELで、このコードを受け取ればよいわけです。

ここでは、次の3つのファイルが必要ですので、確認してください。

- **MASM. EXE**（プログラムをマシン語ファイルにする）
- **LINK. EXE**（マシン語ファイルを実行可能な**EXE**ファイルにする）
- **EXE2BIN. COM**（**EXE**ファイルを**COM**ファイルにする）

アセンブリ言語での処理内容

アセンブリ言語での処理は次のようにします。

- キーボードから入力を受け付け、その内容を表示する。
- 入力されたものが「Y」「N」以外であれば、再び入力させる。

・入力された「Y」「N」に対応して、コードを0,1と出し、バッチ・ファイルに与える。

これをBASIC言語で表現すると、次のようになります。

■アセンブリ言語の機能をBASICで表現

```
1     ' save "YESNO.BAS"
100 '------------------------        ┐
110 ' YESNO BASIC PROGRAM            │ コメント
120 '------------------------        ┘
130 *START
140 AL$=INPUT$(1):PRINT AL$          …1文字入力と表示
150 '------------------------
160 IF AL$="Y" THEN GOTO *YES        ┐「Y」、「y」の判定とジャンプ
170 IF AL$="y" THEN GOTO *YES        ┘
180 IF AL$="N" THEN GOTO *NO         ┐「N」、「n」の判定とジャンプ
190 IF AL$="n" THEN GOTO *NO         ┘
200 GOTO *START                      …1文字入力に戻る
210 '------------------------
220 *YES                             ┐
230   AL=0                           │ 0を表示する用意をする
240 GOTO *BACK                       ┘
250 '------------------------
260 *NO                              ┐ 1を表示する用意をする
270 AL=1                             ┘
280 '------------------------
290 *BACK                            ┐ 値を表示
300 PRINT AL                         ┘
310 END                              …プログラム終了
```

BASIC言語では、Y／Nに応じて0と1を表示するようにしています。その実行結果は、次のようになります。

■BASICでの実行例

```
run
Y    ┐ 「Y」を入力すると
 0   ┘    0を表示
Ok
run
N    ┐ 「N」を入力すると
 1   ┘    1を表示
Ok
```

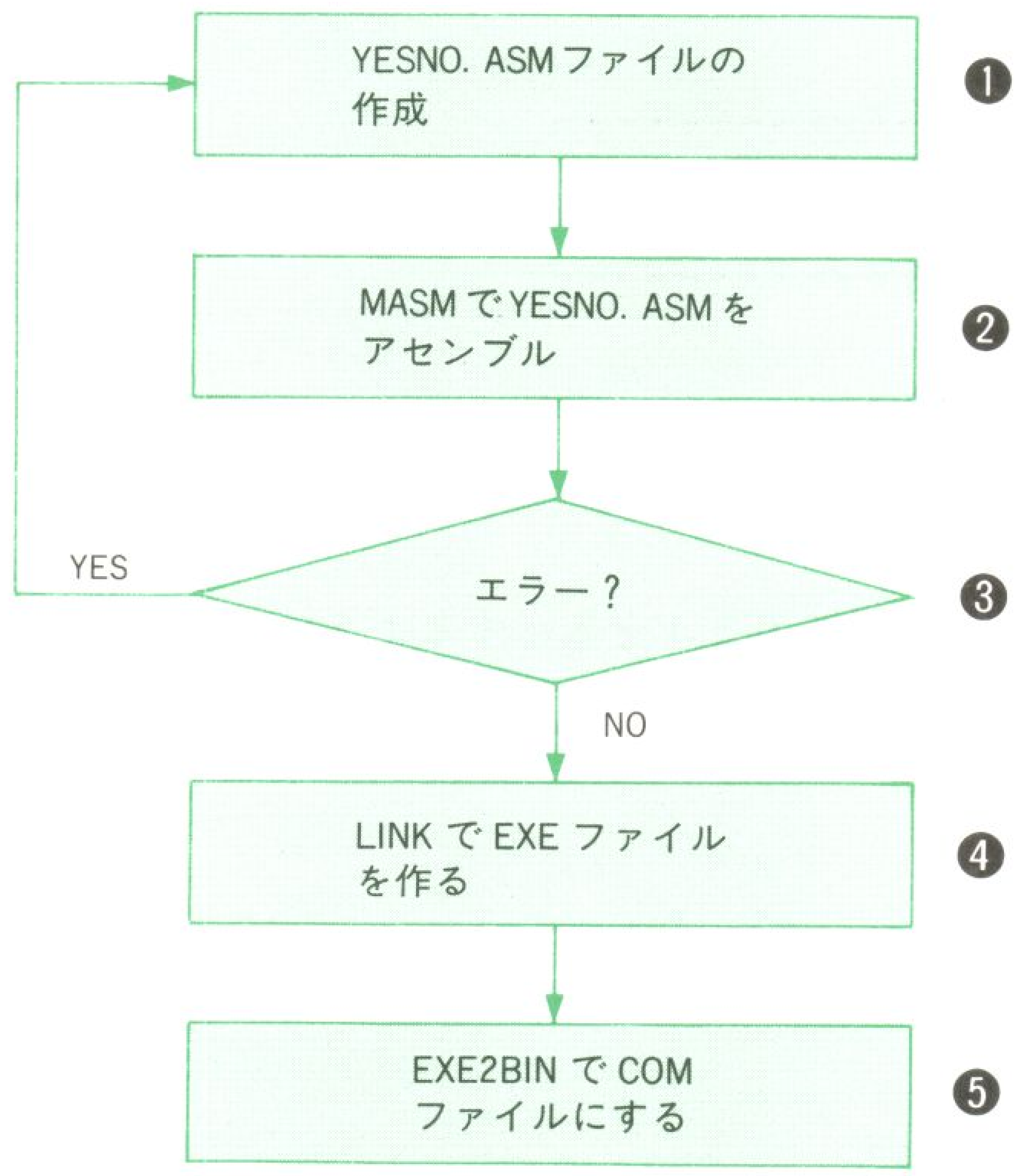
3・2 マシン語のプログラム開発手順

マシン語のプログラムは、次のようにして作ります。

1. エディタでアセンブリ言語のプログラム・ファイルを作る。ファイル名は、YESNO.ASM とする。
2. MASM で YESNO.ASM をアセンブルする。
3. エラーがあれば1に戻って、修正する。
4. エラーがなければ、LINK で EXE ファイルを作る。
5. EXE ファイルを EXE2BIN で COM ファイルにする。

これを図で表わすと、次のようになります。

■マシン語プログラム開発手順



以下、具体的な手順を説明します。

1. ファイル名を YESNO.ASM として、プログラム・ファイルを次のように作ります。とりあえず、このとおり入力してください。右側のセミコロン（；）以降は、アセンブリ言語の処理を BASIC 言

語でコメントとして付けていますので、省略しても結構です。

■アセンブリ言語のプログラム(YESNO. ASM)

```
;-------------------------              ]  コメント
;    YESNO.ASM
; YES OR NO Input
; for Batch file
;-------------------------
CODE SEGMENT                            ]  プログラムの開始宣言
     ASSUME CS:CODE,DS:CODE
     ORG 100H
;---------------------------------
START:                                  ]  1文字入力と表示
     MOV AH,1      ; AL$=INPUT$(1)
     INT 21H       ; PRINT AL$
;-----------------------------------
     CMP AL,'Y'    ; IF AL$="Y" THEN    ]  「Y」,「y」の判定とジャンプ
     JE YES        ; GOTO YES
     CMP AL,'y'    ; IF AL$="y" THEN
     JE YES        ; GOTO YES
;-----------------------------------
     CMP AL,"N"    ; IF AL$="N" THEN    ]  「N」,「n」の判定とジャンプ
     JE NO         ; GOTO NO
     CMP AL,"n"    ; IF AL$="n" THEN
     JE NO         ; GOTO NO
;-----------------------------------
     JMP START     ; GOTO START         ]  1文字入力に戻る
;-----------------------------------
YES:                                    ]  0を返す用意をする
     MOV AL,0      ; AL=0
     JMP BACK      ; GOTO BACK
;-----------------------------------
NO:                                     ]  1を返す用意をする
     MOV AL,1       ; AL=1
BACK:
     MOV AH,4CH     ; Return AL(0,1)    ]  値を返す
     INT 21H        ; to Batch file
;-----------------------------------
CODE ENDS                               ]  プログラム終了宣言
     END START
```

Return AH

2. 次のようにして YESNO. ASM をマシン語ファイル(OBJ ファイル)にする。

A>MASM　YESNO

正確には A>MASM　YESNO. ASM で、ASM は省略可。

次の画面で、[　　　　] 部を入力します。

■ YESNO. ASMのアセンブル

```
A>MASM YESNO ⏎
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

Object filename [YESNO.OBJ]: ⏎ ――― OBJファイルはYESNO. OBJ
Source listing  [NUL.LST]: ⏎        となる
Cross reference [NUL.CRF]: ⏎

49698 Bytes free

Warning Severe
Errors  Errors
0       0

A>
```

ここでエラーが表示されたら
YESNO. ASMの入力ミスですので
修正して，もう1度MASMを
実行します。

表示されたメッセージの意味については、のちほど説明しますので、ここでは無視してください。ただ、エラーのところが、0 にならず 1 や 2 と数字が表示されたら、それだけ YESNO. ASM の中にエラーがあるということです。同時にエラーの行も表示されますので、YESNO. ASM の該当箇所を修正してください。

ここで、OBJ ファイルが作られているか、確認してみましょう。

■ OBJファイルの確認

```
A>DIR YESNO.* ⏎

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

YESNO     ASM      968  87-07-22  16:42
YESNO     OBJ       76  87-07-22  16:45 ――― OBJファイルが
        2 個のファイルがあります.              作られている
    40960 バイトが使用可能です.
```

3. 次のようにして YESNO. OBJ を EXE ファイルにする。

A＞LINK　YESNO

正確には A＞LINK　YESNO. OBJ で OBJ は省略可。

■ YESNO. OBJのリンク

```
A>LINK YESNO ⏎

Microsoft 8086 Object Linker
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985

Run File [YESNO.EXE]:⏎  ——— EXEファイルはYESNO. EXEとなる
List File [NUL.MAP]:⏎
Libraries [.LIB]:⏎
Warning: no stack segment  ——— 注意が表示される

A>
```

Warning：（警告）で注意事項が表示されますが、COMファイルを作る場合は無視してかまいません。

●この警告はＣＯＭファイルを作るときに必ず表示される。

ここでも、EXE ファイルが作られているか、確認してみましょう。

■ EXEファイルの確認

```
A>DIR YESNO.* ⏎

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

YESNO     EXE       801   87-07-22   16:49  ——— EXEファイルが作られている
YESNO     ASM       968   87-07-22   16:42
YESNO     OBJ        76   87-07-22   16:45
          3 個のファイルがあります.
      39936 バイトが使用可能です.

A>
```

4. 次のようにして YESNO. EXE を COM ファイルにする。

A＞EXE2BIN　YESNO　YESNO.COM

正確には、A＞EXE2BIN　YESNO. EXE　YESNO. COM で、EXE は省略可。

■EXEファイルをCOMファイルに変換

```
A>EXE2BIN YESNO YESNO.COM ⏎

A>DIR YESNO.*⏎

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

YESNO     EXE       801  87-07-22  16:49
YESNO     COM        33  87-07-22  16:51 ――― COMファイルが作られている
YESNO     ASM       968  87-07-22  16:42
YESNO     OBJ        76  87-07-22  16:45
        4 個のファイルがあります.
    38912 バイトが使用可能です.

A>
```

これで、YES／NOを判断するYESNO.COMというコマンド・ファイルができました。YESNOと入力して、実行してみてください。

■YES NO. COMの実行

```
A>YESNO ⏎ ――― YESNO.COMの実行
Y ――――――― Yを入力
A>YESNO ⏎
N ――――――― Nを入力
A>YESNO ⏎
1 ――――――― 1を入力しても受けつけない
```

YかNを押せば、MS-DOSのコマンド待ちに戻ります。その他のキーは、受け付けません。ここで、Y／Nに対して、0,1が帰ってくるか、実験してみましょう。

■機能実験

```
A>YESNO ⏎ ――― YESNO.COMの実行
Y ――― Yを入力
A>IF ERRORLEVEL 0 ECHO YES⏎ ――― 0が帰ってきているかのチェック
YES ――― ちゃんと0が帰ってきている

A>YESNO ⏎
N ――― Nを入力
A>IF ERRORLEVEL 1 ECHO NO⏎ ――― 1が帰ってきているかのチェック
NO ――― ちゃんと1が帰ってきている

A>
```

●ECHOとは、バッチ・ファイル用の命令で、そのあとの文字を表示する働きをする。

0→YES
1→NO

ECHOというのは、バッチ・ファイル用の命令で、そのあとにある文字を表示します。そこで、0のときはYES、1のときはNOと表示させてみたわけです。

IF　ERRORLEVELでちゃんと期待の結果が得られています。ここでは、MS-DOSのコマンド待ちの状態に、0,1が帰ってきています（渡されています）。

では、最終目的である、バッチ・ファイルを実行してみてください。次の例は、YESNOファイルをすべて表示して、YES／NOでは、Nを入力したものです。

■バッチ・ファイルの実行

```
A>FDISP YESNO ⏎

A>ECHO OFF
********** ファイル表示 **********
ドライブBに表示したいディスクを入れてください.
準備ができたらどれかキーを押してください.
--------------------------------------------
YESNO     EXE      801   87-07-22   16:49
YESNO     COM       33   87-07-22   16:51
YESNO     ASM      968   87-07-22   16:42
YESNO     OBJ       76   87-07-22   16:45
処理を続けますか（Y／N）？
N
A>
```

YESNOのファイルをすべて表示

YESNOファイルが表示

Nを入力

3·3 アセンブリ言語プログラムの説明

ここで、簡単にアセンブリ言語プログラムの説明をしておきます。プログラムは大きく、次の4つの部分に分かれています。

1. プログラムの表題部
2. プログラム開始宣言部
3. プログラムそのもの
4. プログラムの終了宣言部

以下、各部について説明します。

1. プログラムの表題部

プログラムの内容などコメントを書く部分で、これはプログラムの実行には関係なく、なくてもかまいませんが、通常はあったほうがわかりやすいものです。

■表題部

```
;-------------------------
;    YESNO.ASM
; YES OR NO Input
; for Batch file
;-------------------------
```

先頭にセミコロン（;）を付けると、それ以降はコメントになります。BASIC 言語の REM や「'」と同じことです。ここでは、表題部の他、プログラムを見やすくするために、コメントの代わりに点線を引いています。

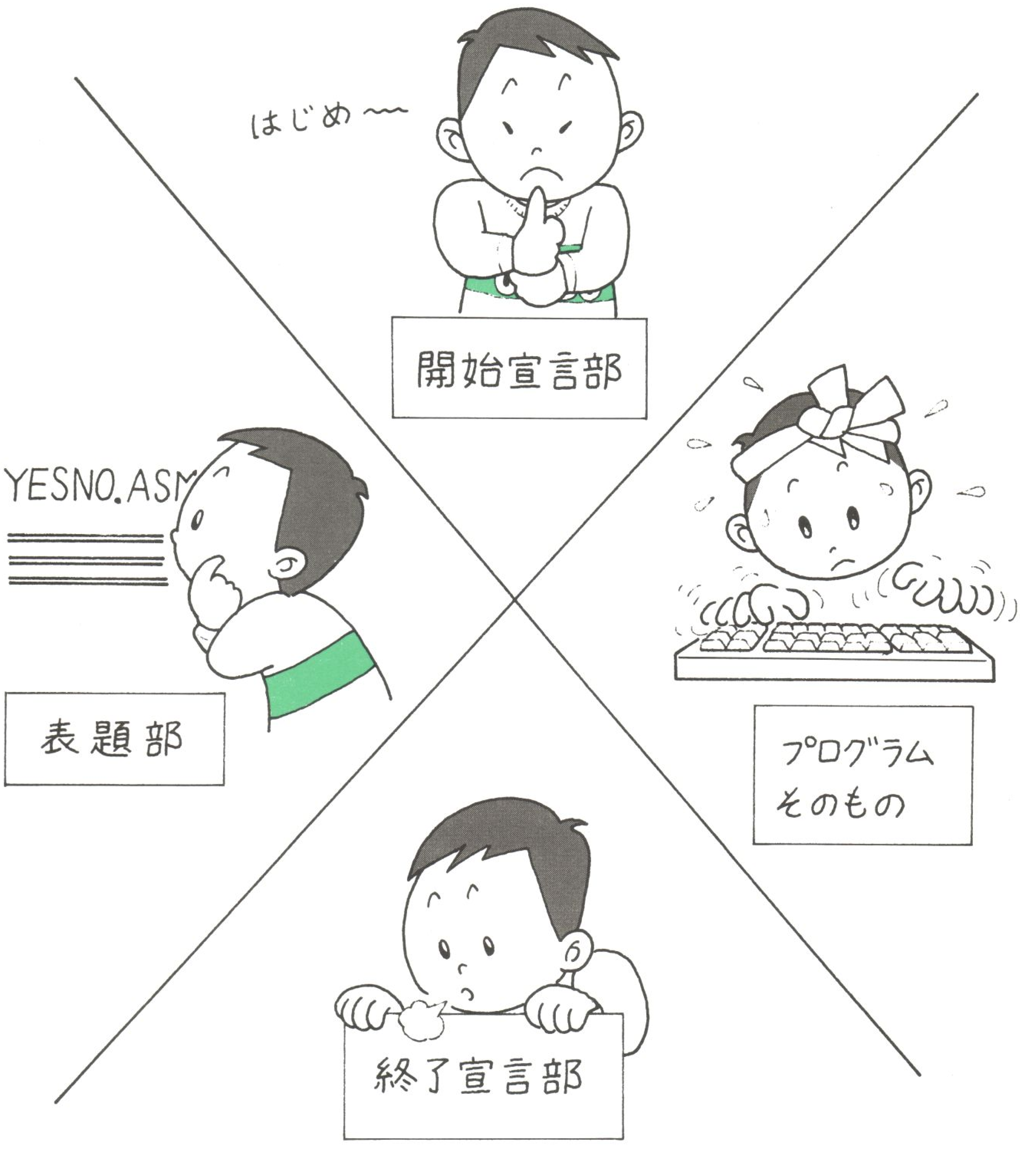
2. プログラム開始宣言部

これは、アセンブラにプログラムの開始位置を伝えるものです。これは、もうおまじないだと思って、この通りに書くものと考えてください（多少はバリエーションがありますが）。

●プログラム開始宣言は、アセンブリ言語ではなく、マクロアセンブラに対するる命令で疑似命令と呼ばれる（「第2部2−3　アセンブラに対する命令」102ページ参照）。

■開始宣言部

```
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
```



3. プログラムそのもの

ここが、マシン語に変換される部分です。プログラムの開始位置や飛び先を示すのにラベルを使っています。基本的には、BASIC 言語のラベルと同じです。

●プログラム中のラベルやクォートに入っている文字は、アセンブラが判読する。

■プログラムそのもの

```
;---------------------------------
START:
     MOV AH,1       ; AL$=INPUT$(1)
     INT 21H        ; PRINT AL$
;----------------------------------
     CMP AL,'Y'     ; IF AL$="Y" THEN
     JE YES         ; GOTO YES
     CMP AL,'y'     ; IF AL$="y" THEN
     JE YES         ; GOTO YES
;----------------------------------
     CMP AL,"N"     ; IF AL$="N" THEN
     JE NO          ; GOTO NO
     CMP AL,"n"     ; IF AL$="n" THEN
     JE NO          ; GOTO NO
;----------------------------------
     JMP START      ; GOTO START
;----------------------------------
YES:
     MOV AL,0       ; AL=0
     JMP BACK       ; GOTO BACK
;----------------------------------
NO:
     MOV AL,1        ; AL=1
BACK:
     MOV AH,4CH      ; Return AL(0,1)
     INT 21H         ; to Batch file
;----------------------------------
```

4. プログラムの終了宣言部

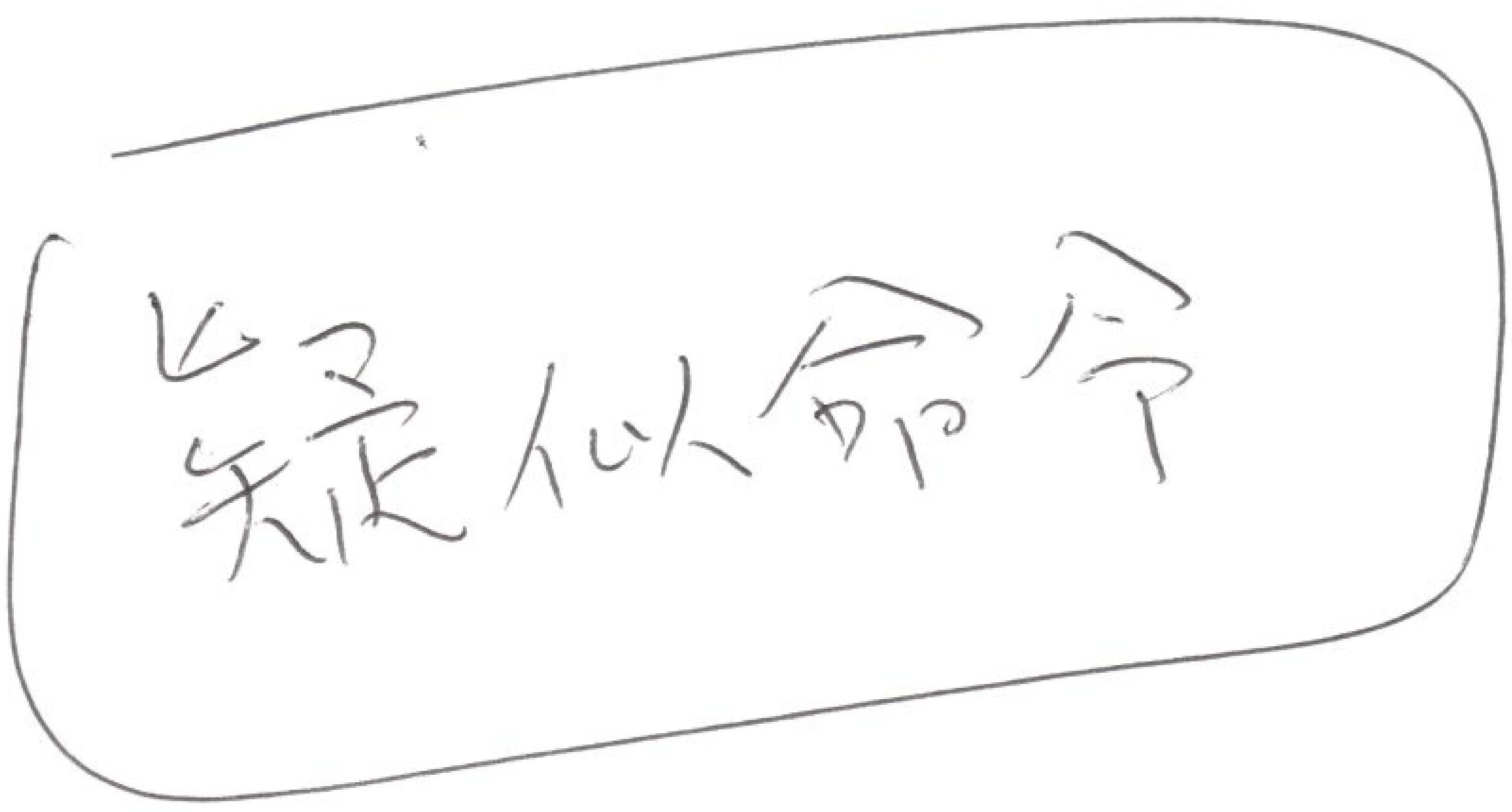
アセンブリ言語でのプログラムの終わりを示すものです。これもアセンブラに伝えるもので、おまじないと思ってください。

●プログラム終了宣言も、アセンブリ言語ではなく、マクロアセンブラに対する命令で疑似命令(「第2部2−3　アセンブラに対する命令」102ページ参照)。

■終了宣言部

```
CODE ENDS
     END START
```

以上のことから、アセンブラ用にアセンブリ言語を書くには、アセンブリ言語そのものの他に、アセンブラに対してアセンブルするときの命令を書く必要があることがわかります。この例では、プログラム開始宣言部と終了宣言部がそれに当たります。これらの命令を本当のアセンブリ言語の命令ではないことから、疑似命令といいます。



3·4 プログラムそのものの説明

MOV AH, 1

意味：AH の内容を 1 にする。AH に 1 を入れる。
MOV：MOVe のことで値を代入する命令
AH　：CPU8086 が内部で持っている変数のような役割を果たすものの 1 つ

INT 21 H

意味：MS-DOS で使う専用命令で、AH の内容により機能が異なる。
INT　：INTerrupt のことで、ソフトウェアによる割り込み処理を実行する 8086 の命令。
21 H：MS-DOS で用意されている機能の大部分が、この割り込み番号で利用される。

```
MOV AH, 1
INT 21 H
```

で、次のような機能となります。

- キーボードから 1 文字入力されるまで待つ。
- ユーザーはキーを 1 つ押すだけでよく、リターンキーを押す必要がない。
- 入力された 1 文字を画面に表示する。
- 入力された 1 文字の文字コード（アスキーコード）が、AL（8086 が内部で持っている変数のような役割を果たすものの 1 つ）に格納される。

これを BASIC 言語で表わせば、

```
AL$=INPUT$(1)：PRINT AL$
```

ということになります。

CMP AL, 'Y'

意味：AL の内容と文字 Y とが同じであるかを比較する命令。この前の命令の実行で、AL にアスキーコードが入っているので、それが Y かどうか比較している。

CMP：CoMPare のことで、比較する 8086 の命令。

'Y'　：クオート（'）で文字をくくると、それがアセンブラでアスキーコードに直される。ユーザーは、Y のアスキーコードを知らなくてもよい。

JE YES

意味：比較の結果、等しいならラベル YES のところに飛ぶ（処理を移す）命令。

JE　：Jump Equal ということで、比較の結果が等しいならジャンプするという 8086 の命令。

YES：その処理の内容を表わしたラベル名。アセンブリ言語の中でユーザーが思うようにつけられる。

以上を BASIC 言語で表わすと、

IF AL$＝"Y"　THEN GOTO ＊YES

という１つの条件文となります。
アセンブリ言語では、まず内容を比較して、その結果、処理を移すというように２つの命令を組み合わせることになります。

CMP AL, 'y'
は、小文字の **y** と **AL** との比較です。

以下、N と n の比較とジャンプも同じことです。

JMP START

意味：ラベル **START** のところに制御を移す（ジャンプする）。
JMP：JuMP のことで、文字どおりジャンプする命令。
START：プログラムの開始位置を示すラベル名。アセンブリ言語の中でユーザーが思うようにつけられる（BEGIN などとしてもよい）。

MOV AL, 0

意味：AL に 0 を代入する命令。

MOV AL, 1

意味：AL に 1 を代入する命令。

これで、YES／NO によって、AL に 0 か 1 かを入れています。

JMP BACK

意味：ラベル BACK にジャンプする命令。

MOV AH, 4CH

意味：AH に 16 進数で 4C を代入する命令。4CH の H は Hexadecimal（16 進数）の意。

```
MOV  AH, 4CH
INT  21H
```

で、AL の内容をバッチ・ファイルに与える機能となります。

冒頭で、

MOV AH, 1
INT 21H

として、1文字入力の機能を利用しましたが、ここでは、AHに4CHを入れて、ALの内容をバッチ・ファイルに与える機能を利用しています。このようにAHの値を変えて、INT　21Hを実行することにより、MS-DOSで用意されているいろいろな機能が利用できます。

● INT 21Hには、アセンブリ言語でプログラムを開発するのにユーザーが利用できる機能が集められている。これをファンクション・コール(機能呼び出し)という。

第2部 マシン語とアセンブラを理解する

第2部では、第1部で体験したマシン語とアセンブラをベースにして、本格的な説明をします。マシン語学習に最低限知っておきたいことがらとマクロアセンブラについて解説したあと、アセンブリ言語でのプログラミング力を付けるために、第1部で学んだプログラムを拡張してみます。これにより、第1部でぼんやりしていたことがはっきりとみえてきて、マシン語とアセンブラの理解が深まることでしょう。

SEM

第2部

第1章 最低限、知っておくこと

この章では、マシン語を学ぶうえで最低限知っておきたいことがらについて解説します。コンピュータの基本動作から、マシン語とデータを格納するメモリとその位置を示すアドレス、16進数での数の数え方、8086の構造と機能までを取り上げます。

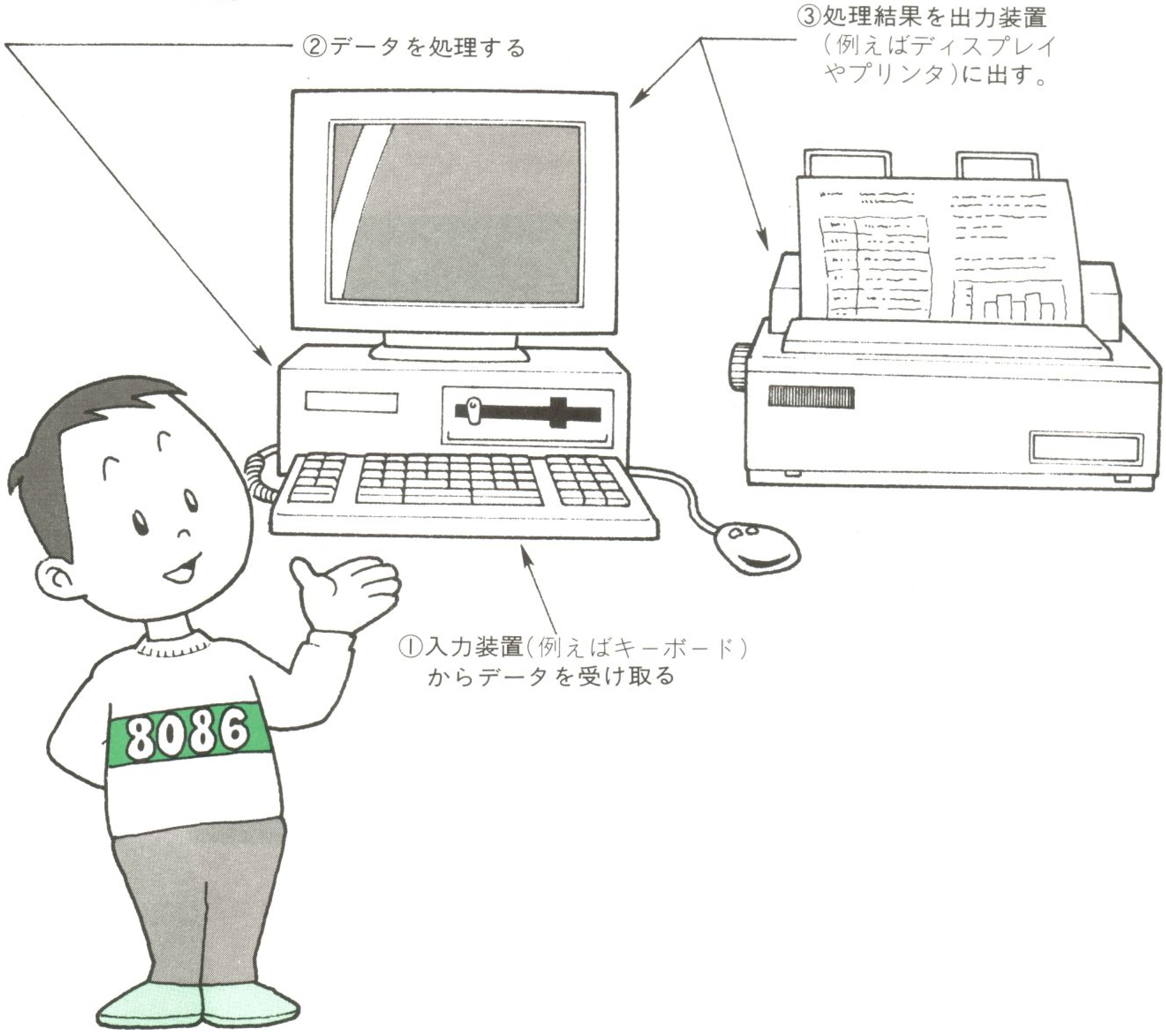
1·1 コンピュータの基本動作

コンピュータの役割は次の3つからなっています。

1. 入力装置からデータを受け取る。
2. そのデータを処理する。
3. 最終処理結果を出力装置に出す。

これを図示すると、次のようになります。

■コンピュータの役割

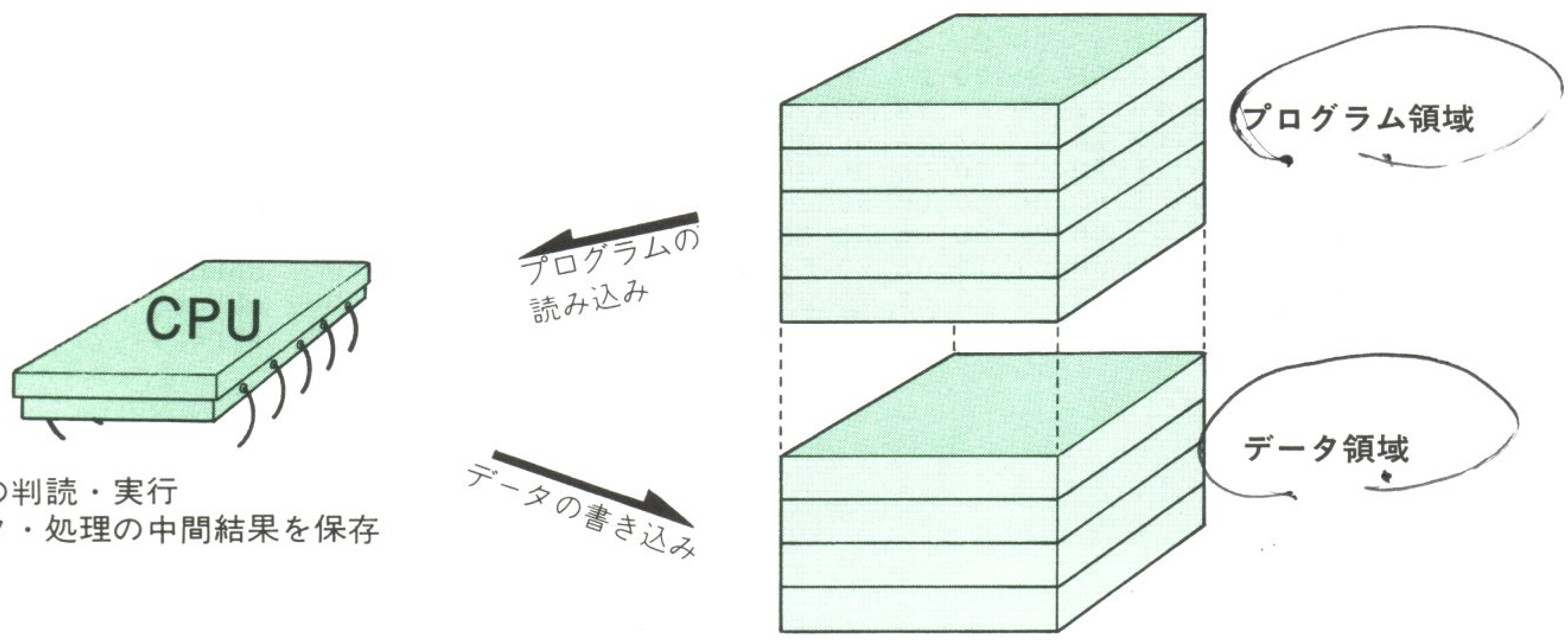


処理の内容は、「プログラム」と呼ばれる一連の命令によって規定されています。プログラムを記憶する場所を「プログラム領域」といいます。また、処理するデータを記憶する場所を「データ領域」といいます。プログラム領域とデータ領域は、どちらもメモリにとられます。**データおよび処理の中間結果はCPU内のメモリ（レジスタという）にも、一時記憶されます。**CPU、プログラム領域、データ領域の関係を示したのが、次の図です。

●プログラム領域は、メモリ内でプログラムを入れるところ。

●データ領域は、メモリ内でデータを入れるところ。

■プログラム領域とデータ領域



なお、メモリのどの部分がプログラム領域となり、どの部分がデータ領域となるかは、プログラマが指定したり、アセンブラ、リンカが自動的に割り当てたりして決定されます。しかしいずれにせよ、メモリのどの部分がデータ用とか、あらかじめ決まっているわけではありません。

第１部第２章のファンクションキーの内容を読み取るプログラムでは、次のようにプログラム領域とデータ領域がとられています。

プログラム領域

```
-U100,109
1A10:0100 BA0002         MOV   DX,0200
1A10:0103 B10C           MOV   CL,0C
1A10:0105 B80000         MOV   AX,0000
1A10:0108 CDDC           INT   DC
-D200,31F
1A10:0200  FE 20 43 31 20 20 1B 53-00 00 00 00 00 00 00 00   ~ C1   .S.....
1A10:0210  FE 20 43 55 20 20 1B 54-00 00 00 00 00 00 00 00   ~ CU   .T.....
1A10:0220  FE 20 43 41 20 20 1B 55-00 00 00 00 00 00 00 00   ~ CA   .U.....
1A10:0230  FE 20 53 31 20 20 1B 56-00 00 00 00 00 00 00 00   ~ S1   .V.....
1A10:0240  FE 20 53 55 20 20 1B 57-00 00 00 00 00 00 00 00   ~ SU   .W.....
1A10:0250  FE 56 4F 49 44 20 1B 45-00 00 00 00 00 00 00 00   ~VOID  .E.....
1A10:0260  FE 4E 57 4C 20 20 1B 4A-00 00 00 00 00 00 00 00   ~NWL   .J.....
1A10:0270  FE 49 4E 53 20 20 1B 50-00 00 00 00 00 00 00 00   ~INS   .P.....
1A10:0280  FE 52 45 50 20 20 1B 51-00 00 00 00 00 00 00 00   ~REP   .Q.....
1A10:0290  FE 20 5E 5A 20 20 1B 5A-00 00 00 00 00 00 00 00   ~ ^Z   .Z.....
1A10:02A0  64 69 72 20 61 3A 0D 00-00 00 00 00 00 00 00 00   dir a:........
1A10:02B0  64 69 72 20 62 3A 0D 00-00 00 00 00 00 00 00 00   dir b:........
1A10:02C0  63 6F 70 79 20 00 00 00-00 00 00 00 00 00 00 00   copy ........
1A10:02D0  64 65 6C 20 00 00 00 00-00 00 00 00 00 00 00 00   del .........
1A10:02E0  72 65 6E 20 00 00 00 00-00 00 00 00 00 00 00 00   ren .........
1A10:02F0  63 68 6B 64 73 6B 20 61-3A 0D 00 00 00 00 00 00   chkdsk a:....
1A10:0300  63 68 6B 64 73 6B 20 62-3A 0D 00 00 00 00 00 00   chkdsk b:....
1A10:0310  74 79 70 65 20 00 00 00-00 00 00 00 00 00 00 00   type ........
-
```

プログラム

データ領域

データ

コンピュータの処理は、「制御装置」という部分が、次の3つのステップを繰り返して、行われます。

1. 命令の取り出し
2. 命令の判読
3. 命令の実行

各ステップは、次のような処理をします。

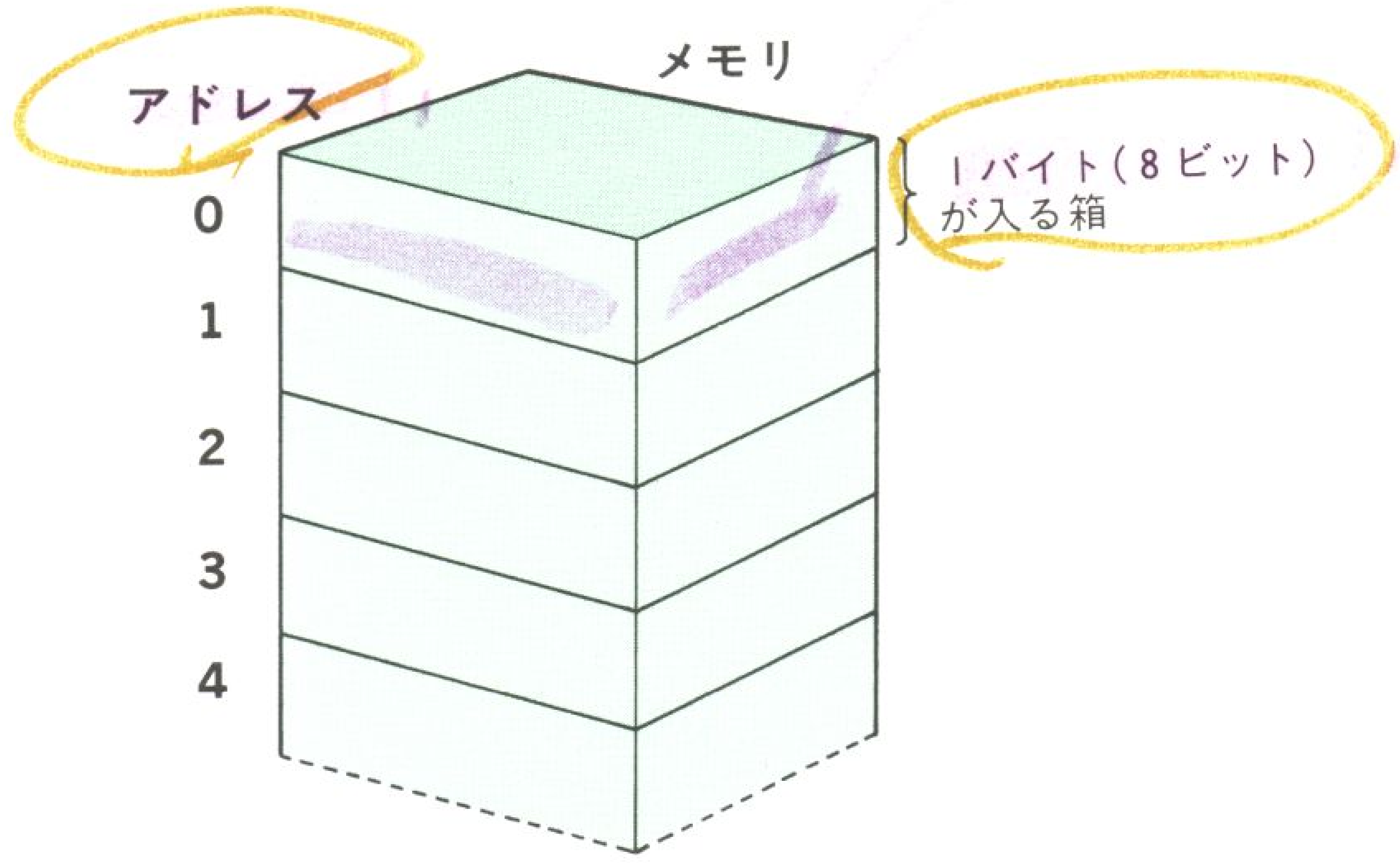
1. まず、プログラム領域から命令を1つ取ってくる。
2. 次に、その命令を判読し、どういう動作をするか決める。
3. そして、その動作を受け持つ装置に制御信号を送り、命令を実行する。

1・2 メモリとアドレス

　メモリとは、定められた情報の単位が入るひと続きの場所の集まりです。そして、それぞれの場所に「アドレス」(Address：番地) が付けられています。1つの場所には、普通1バイト (8ビット) の情報が入ります。

　メモリとアドレスの様子を示したのが、次の図です。通常、アドレスは0から始まります。そして、16進数で表わされます(16進数については、1−3を参照)。

■メモリとアドレス



　つまり、メモリとは、1バイトが入る箱の集まりで、その箱の1つ1つを識別するために付けられた番号がアドレスです。ファンクションキーの内容を読み取るプログラムでは、次のようにプログラムがアドレス100から109までに格納されています。

■プログラムの格納のされ方とアドレス

```
-
-U100,109
1A10:0100 BA0002        MOV    DX,0200
1A10:0103 B10C          MOV    CL,0C
1A10:0105 B80000        MOV    AX,0000
1A10:0108 CDDC          INT    DC
-D100,109
1A10:0100  BA  00  02  B1  0C  B8  00  00-CD  DC                    :..1.8..M¥
           100 101 102 103 104 105 106 107 108 109
-
-
```

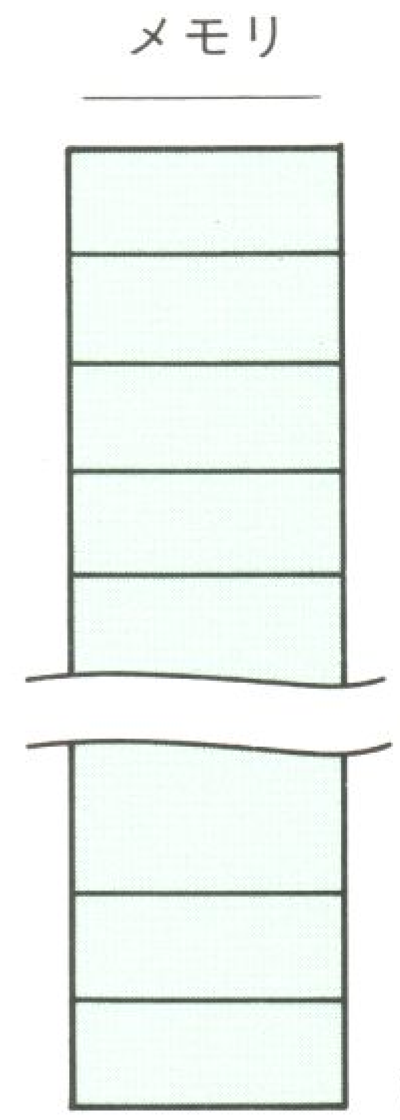
1バイトは、2桁の16進数で表わされます。なお、実際の格納アドレスは、コロン（：）の左側のアドレスとあわせて表わされます（1−4「セグメントの概念」を参照）が、ここでは無視して、コロン（：）の右側のアドレスだけを見てください。

1·3 16進数

8086のシステムでは1バイトのメモリをひと続きに1メガ（M）バイトまでアクセス（読み書き）することができます。これをアドレスで表現すると、次のようになります。

■1Mバイトのメモリとアドレス

16進表現アドレス	2進表現アドレス
00000	0000 0000 0000 0000 0000
00001	0000 0000 0000 0000 0001
00002	0000 0000 0000 0000 0010
00003	0000 0000 0000 0000 0011
⋮	⋮
FFFFE	1111 1111 1111 1111 1110
FFFFF	1111 1111 1111 1111 1111



0と1の2進数では、

0000 0000 0000 0000 0000

から

1111 1111 1111 1111 1111

までとなります。

16進数では、

00000

から

FFFFF

までとなります。

16 進数のほうが、表現が簡単で見た目もわかりやすいようです。マシン語開発を行うときは、数値やアドレスを通常 16 進数で表わします。アセンブリ言語をマシン語に直したときも、16 進数で表わされます。これは、マシン語の世界では、16 進数で表わすのが、なにかと便利だからです。

アセンブラを使ってマシン語を開発するときには、10 進数で表現することもできます。アセンブラがアセンブリ言語をマシン語にするときに 10 進数を 16 進数に変えるからです。しかし、16 進数での数の数え方をマスターしておいたほうが、マシン語やアセンブリ言語の理解が早まります。

そこで、ここで 16 進数での数の数え方を学びましょう。私たちの普段の生活では、10 進数を使っていますが、これは、

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

という 10 個の数を使って、数が 10 個集まれば、1 ケタ繰り上げる数え方をしています。10 以上は、次のように表現しています。

10, 11, 12, 13, 14, 15. . .

● 16 進数は、0−9,A−F の 16 個の数を使って表現する。

16 進数も同じことで、

0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
A, B, C, D, E, F

という 16 個の数を使って、数が 16 個集まれば、1 ケタ繰り上げる数え方をします。アラビア数字は 10 個までしかないため、アルファベットの A から F までを数として取り扱っています。10 進数と対応させると、次のようになります。

10 進	16 進
10	A
11	B
12	C
13	D
14	E
15	F

16 以上は、次のように表現します.

10 進	16 進	10 進	16 進
16	10	25	19
17	11	26	1A
18	12	27	1B
19	13	28	1C
20	14	29	1D
21	15	30	1E
22	16	31	1F
23	17	32	20
24	18	33	21

16 進では、9 と 10 との間に A, B, C, D, E, F が入っていると覚えてください。9 の次は A、F の次が 10 と数えてください。

16 進数は、英語で Hexadecimal(ヘキサデシマル)といいますので、アセンブリ言語では、16 進数を表わすのに、その後ろに H を付けます。

ただし、先頭が 0~9 の数字でなく、A~F で始まる場合 (A0H など) には、その前に 0 を付けます (0A0H)。

●0 を付けなければ、アセンブラは 16 進数を文字としてみなします。

例：1AH（10 進で 26）
　　1BH（10 進で 27）
　　0FFFFH（10 進で 65535）

これは、BASIC 言語で 16 進数を表わすのに、&H を付けるのと同じことです。

例：&H1A（10 進で 26）
　　&H1B（10 進で 27）

なお、16進数では、2桁で1バイト（8ビット）を表わし、

00, 01, . . . , 0A, 0B, . . . , 1F

などと普通書きます。これが、2バイトになれば、

0000, 0001, . . . 000A, 000B
0200, 3FFF, . . . FFFE, FFFF

などと表わします。

ちなみに、覚えておくと便利な10進・16進対応表を次に紹介します。

■覚えておくと便利な10進・16進対応

10進数	16進数
10	0A
16	10
32	20
64	40
128	80
255	0FF
256	100
512	200
1024	400
2048	800
65535	0FFFF

■ 16進数を実際に見る

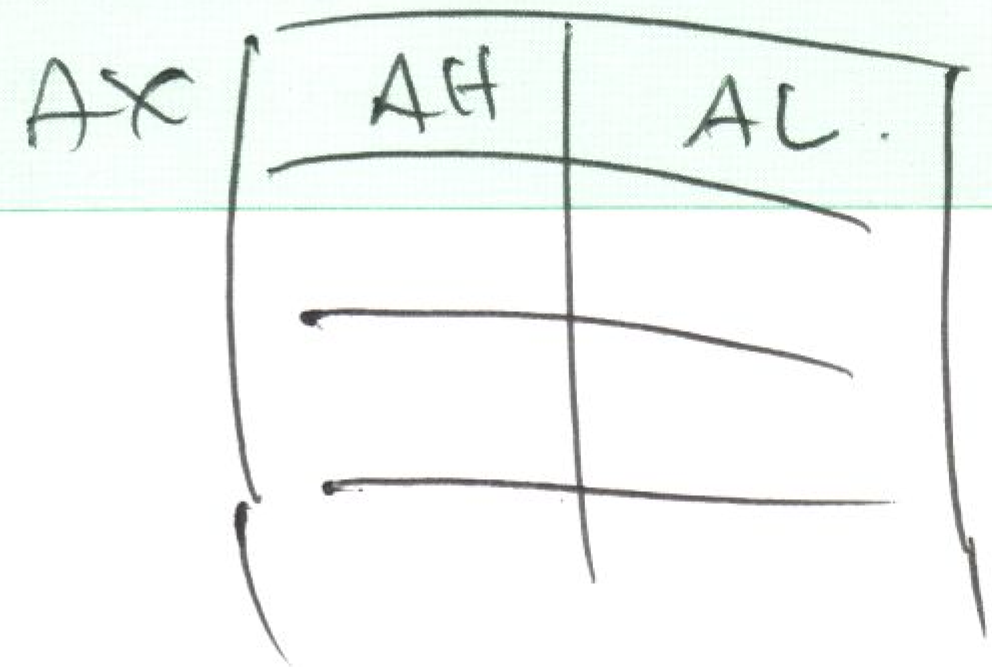
ここで、16進数を実際に見てみることにしましょう。マシン語プログラムを実行して、読者が使っている MS-DOS のバージョン番号を調べると、それが16進数で示されます。

デバッガ（SYMDEB または DEBUG）を起動して、次のように入力してください。

■バージョン3.10でバージョン番号を見る

```
A>SYMDEB ⏎
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-A ⏎
1A10:0100 MOV AH,30 ⏎ ┐── バージョン番号を得る
1A10:0102 INT 21 ⏎    ┘      プログラム
1A10:0104 ⏎
-G104 ⏎ ── 実行
AX=0A03  BX=FF00  CX=0000  DX=0000  SP=E4DF  BP=0000  SI=0000  DI=0000
DS=1A10  ES=1A10  SS=1A10  CS=1A10  IP=0104   NV UP EI PL NZ NA PO NC
1A10:0104 0000          ADD    [BX+SI],AL                        DS:FF
-RAX ⏎ ── AXの内容表示
AX 0A03
:                ── 3.10
-Q ⏎

A>
```



■バージョン2.11でバージョン番号を見る

```
A>DEBUG ⏎
-A ⏎
1355:0100 MOV AH,30 ⏎ ┐── バージョン番号を得る
1355:0102 INT 21 ⏎    ┘      プログラム
1355:0104 ⏎
-G104 ⏎ ── 実行

AX=0B02  BX=FF00  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1355  ES=1355  SS=1355  CS=1355  IP=0104   NV UP DI PL NZ NA PO NC
1355:0104 2442          AND     AL,42
-RAX ⏎ ── AXの内容表示
AX 0B02 ── 2.11
:
-Q ⏎

A>
```

これは、MS-DOSのバージョン番号を取り出すプログラムです。

MOV AH, 30
INT 21H

とすると、8086の変数のようなもの（レジスタ）AXに、16進数でバージョン番号が入るようになっています。実行後AXの内容が自動的に表示されていますが、確認のため

－RAX

として、AXの内容を表示しています（RはRegisterの内容を表示するコマンド）。

バージョン3.10では、

0A03

となっています。

これは、03が整数部の3を、0Aが小数部の10を表わしています（前出の10進・16進対応を参照）。

バージョン2.11では、

0B02

となっています。

これは、02が整数部の2を、0Bが小数部の11を表わしています（前出の10進・16進対応を参照）。

1・4 8086の構造

■レジスタとは

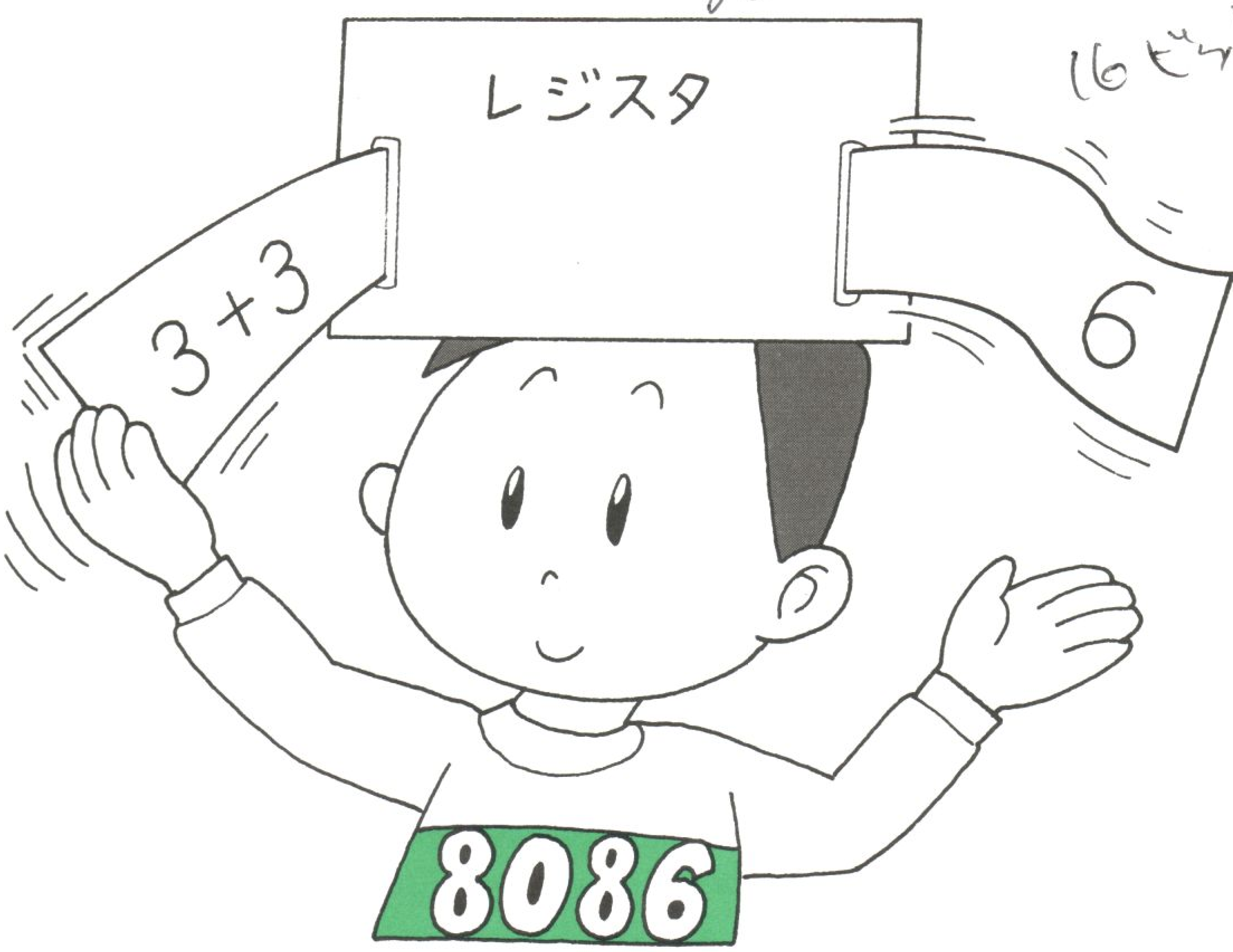
SYMDEB (DEBUG) でマシン語を実行したときに、AX, BX, CX などが表示されましたが、これらは、レジスタ(Register：記録表示器)と呼ばれるものです。

レジスタは、CPUの中にあり、データや演算の結果などを一時的に記憶しておく場所です。メモリに比べてアクセスが簡単で速いという特長があります。レジスタには、直接、値を代入したり、メモリから内容を取ってきて入れたりすることができます。このレジスタを使って、四則演算や論理演算ができます。

BASIC言語で使用する変数のような役割を果たしますがまったく同じではありません。BASIC言語では、変数はユーザーが自由に名前を付け、いくつでも使うことができますが、レジスタの名前と数、機能はCPUにより決められています。

●レジスタは、変数のような役割を果たすが、その名前、数、機能はあらかじめ決められている。

8086では、次のように14のレジスタがあります(8086は、16ビットCPUなので、1つのレジスタは、16ビットの情報を記録できます)。



■8086のレジスター覧

グループ	レジスタ	名称
①データ処理レジスタ・グループ	AX	AX (Accumulator) アキュムレータ
	BX	BX (Base register) ベース レジスタ
	CX	CX (Count register) カウントレジスタ
	DX	DX (Data register) データレジスタ
②アドレス指定レジスタ・グループ	SP	SP (Stack Pointer) スタック ポインタ
	BP	BP (Base Pointer) ベース ポインタ
	SI	SI (Source Index) ソース インデックス
	DI	DI (Destination Index) デスティネーションインデックス
③セグメント指定レジスタ・グループ	CS	CS (Code Segment) コード セグメント
	DS	DS (Data Segment) データ セグメント
	SS	SS (Stack Segment) スタック セグメント
	ES	ES (Extra Segment) エキストラセグメント
④命令ポインタ・レジスタ（実行手順を示す）	IP	IP (Instruction Pointer) インストラクションポインタ
⑤フラグ・レジスタ（状態を示す）	FLAGS	Flags（フラグ）

これらが、これまでデバッガでプログラムを実行したり、R(レジスタ表示コマンド）を実行したときに表示されていたものです。

■8086のレジスタ（デバッガでの表示）

```
A>SYMDEB ⏎
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-R ⏎ ―― レジスタを表示
AX=0000  BX=0000  CX=0000  DX=0000  SP=E4DF  BP=0000  SI=0000  DI=0000
DS=1A10  ES=1A10  SS=1A10  CS=1A10  IP=0100   NV UP EI PL NZ NA PO NC
1A10:0100 0000          ADD     [BX+SI],AL                       DS:00
-
                                    └── フラグ
```

このレジスタ一覧と１つ違うのは、フラグの表示ですが、これはのちほど、フラグのところで説明します。

これらのレジスタは、機能別に、次の３つのグループと２つの特殊レジスタに分類できます。

1. データ処理レジスタ・グループ
2. アドレス指定レジスタ・グループ
3. セグメント指定レジスタ・グループ
4. 命令ポインタ・レジスタ
5. フラグ・レジスタ

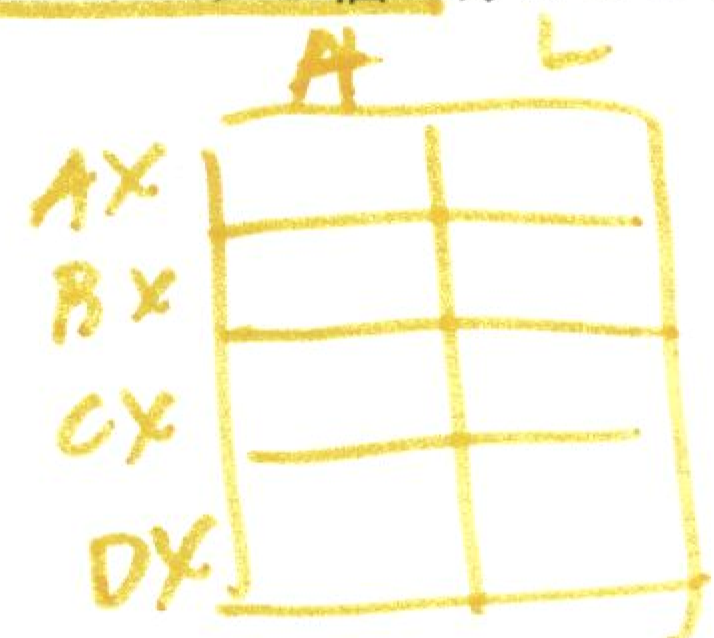
以下、これらのレジスタの特徴と機能を簡単に説明します。

1. データ処理レジスタ・グループ

データの四則演算、掛算、割り算、論理演算を主に行うレジスタです。AX, BX, CX, DX ともによく使います。これらの特徴は、16 ビットのレジスタ１個を８ビットのレジスタ２個に分けることができることです。

● 16 ビットのレジスタの内容を単純に８ビット２つにわけたり、ひっつけたりするということはレジスタ特有の機能です。

AX → AH, AL
BX → BH, BL
CX → CH, CL
DX → DH, DL



2. アドレス指定レジスタ・グループ

レジスタのデータをメモリに格納したり、メモリからレジスタにデータを入れるときに、アドレスを指定する必要がありますが、このアドレスを入れるレジスタです。

● 次のセグメントの概念参照。

3. セグメント指定レジスタ・グループ

アドレスを指定する前に、どのセグメントを選ぶかという大前提があります。このセグメントを入れるレジスタです（セグメントについては後述）。

4. 命令ポインタ・レジスタ

命令を実行するアドレスを指し示すもので、プログラム領域で実行する命令のアドレスが自動的に入ります。

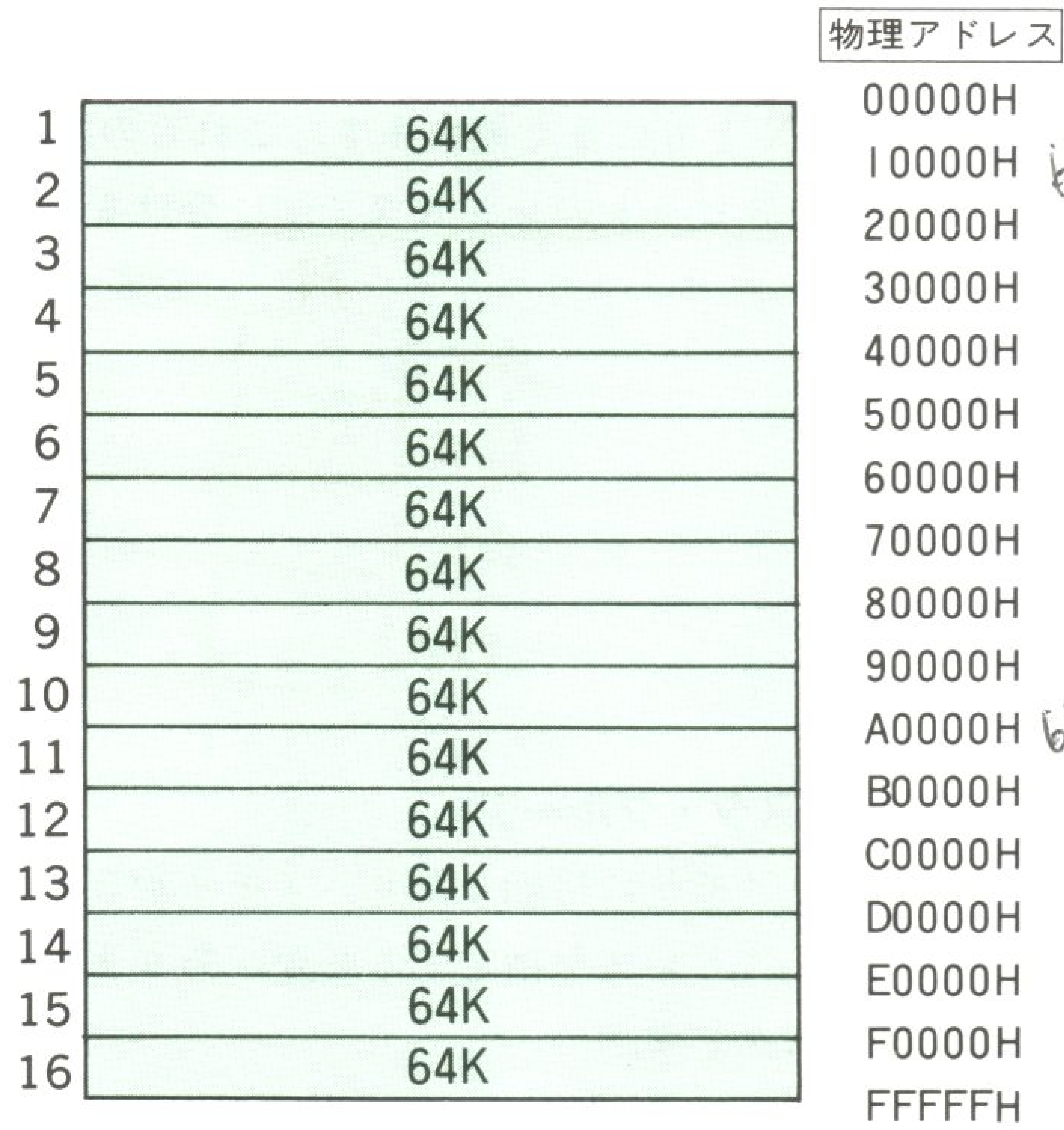
5. フラグ・レジスタ

演算結果や状態を示すレジスタです。処理に応じて自動的に結果や内容が入ります。

■セグメントの概念

8ビットのCPUでは、最大64K(キロ：1Kバイト＝1,024バイト)バイトのメモリしかアクセスできませんが、8086では、1M(メガ：1Mバイト＝1K×1Kバイト) バイトのメモリをアクセスすることができます。これは、次のように64Kバイトの16倍のメモリです。

■1Mバイトのメモリ



●20ビットとは、16進数で4ビットを1桁で表わすので、5桁の16進数となる。16ビットのレジスタでは、4桁までしか表現できない。

1Mバイトのメモリをアドレスで表現するには、20ビットが必要です。しかし、8086では16ビットのレジスタしか持っていません。そこで、16ビットのレジスタを2つ利用して、20ビットのアドレスを表現するという方法をとっています。

1つのレジスタでは、最大64Kバイトのメモリアクセスができます。そこで、セグメント(Segment：区分)という考え方を取り入れて1Mバイトのメモリを1つのレジスタで64Kバイトごとに区分し、その区分した中で実際に読み書きするアドレス(これを物理アドレスという)を指定するという方法をとっています。

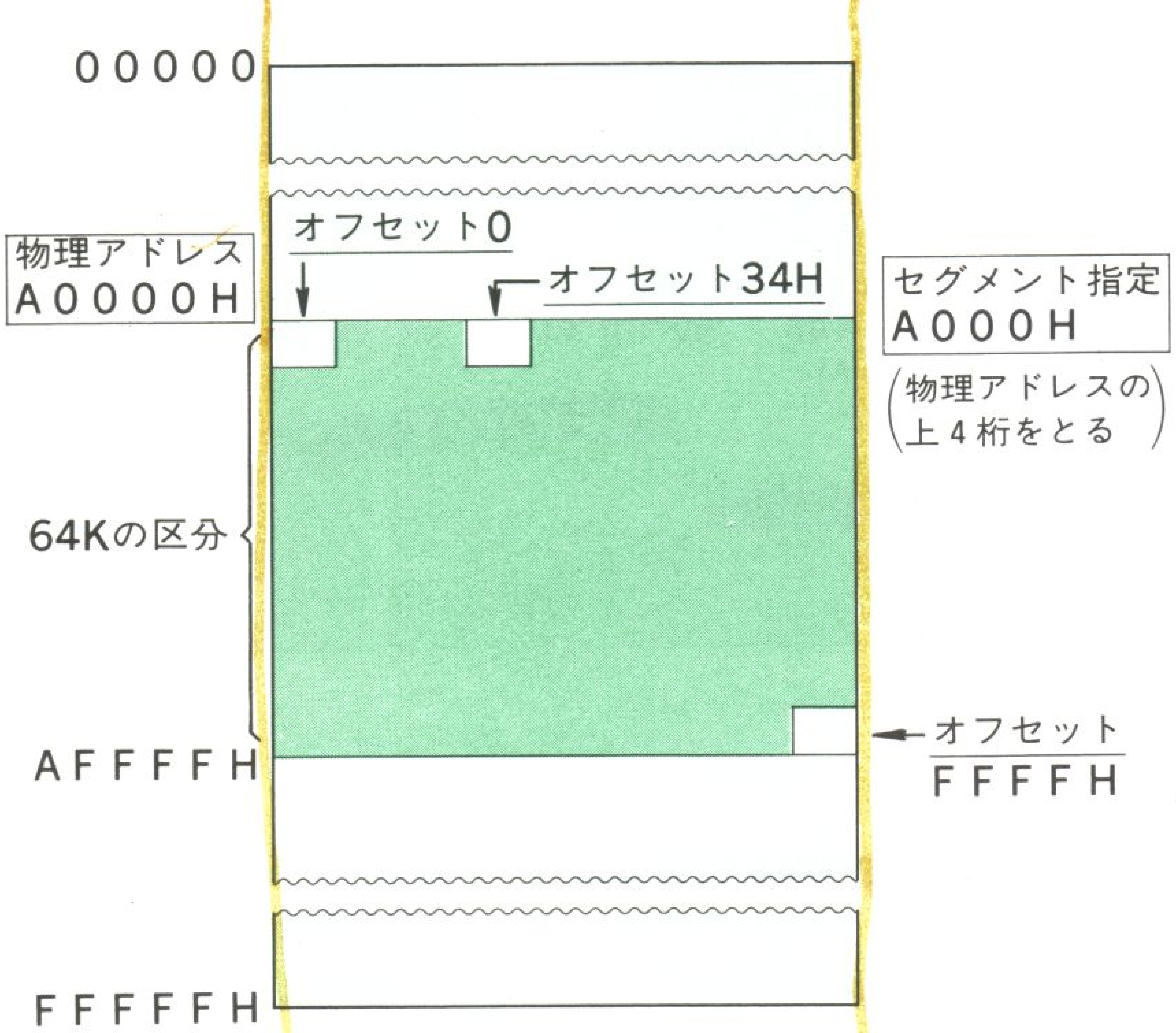
●物理アドレスは、メモリ上の実際の位置を示す。

物理アドレス
セグメントアドレス：オフセットアドレス.

この区分したアドレスをセグメント・アドレスといい、その中でのアドレスをオフセット（Offset：補う）・アドレスといいます。このアドレスの指定方法を図で表わすと、次のようになります。

●物理アドレスは、セグメント・アドレスとオフセット・アドレスを用いて表現する。

■セグメントとオフセットの指定



物理アドレスA0034Hを表すには、まずセグメントをA000Hと指定し、次にオフセットを34Hとする。（セグメントは物理アドレスの上4桁をとって指定すればよい）

●物理アドレスA0034Hの表わし方は次のとおり。

セグメント:A000H
オフセット:0034H

これは、次のようにしてCPUが計算する。

A0000H
＋0034H
A0034H

セグメント・アドレスを入れるレジスタが、セグメント・レジスタで、オフセット・アドレスを入れるレジスタが、アドレス指定レジスタ・グループのレジスタです。

デバッガでのアドレス表示は、実はセグメント・アドレスとオフセット・アドレスとを組み合わせたもので、コロン（：）の左側のアドレスがセグメント・アドレスで、コロン（：）の右側のアドレスがオフセット・アドレスです。

●ファンクションキーの読み込みで、1A10:100とあったのは、セグメント・アドレスが1A10でオフセット・アドレスが100ということ。この場合、物理アドレスは、次のようにして計算される。

1A100H
＋　100H
1A200H

セグメントアドレス：オフセットアドレス

■デバッガでのアドレス表示

セグメント　　オフセット

```
-
-
-D100,16F
1A10:0100  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1A10:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1A10:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1A10:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1A10:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1A10:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1A10:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=E4DF  BP=0000  SI=0000  DI=0000
DS=1A10  ES=1A10  SS=1A10  CS=1A10  IP=0100   NV UP EI PL NZ NA PO NC
1A10:0100 0000          ADD     [BX+SI],AL
-
-
```

オフセット（アドレスは 100H）

セグメント・レジスタ（アドレスはすべて 1A10H）

■ソフトウェア割り込み

割り込みとは、ある処理をしている最中に、それをストップして（それに割り込んで）、別の処理を行うといったことをいいます。そしてその別の処理が終わったら、もとの処理を続けることになります。

仕事中に机の電話が鳴ることも「割り込み」といえます。電話が鳴ったら仕事を中断して、電話をとり、話さなければなりません。そして話が終わったら、中断していた仕事を続けます。

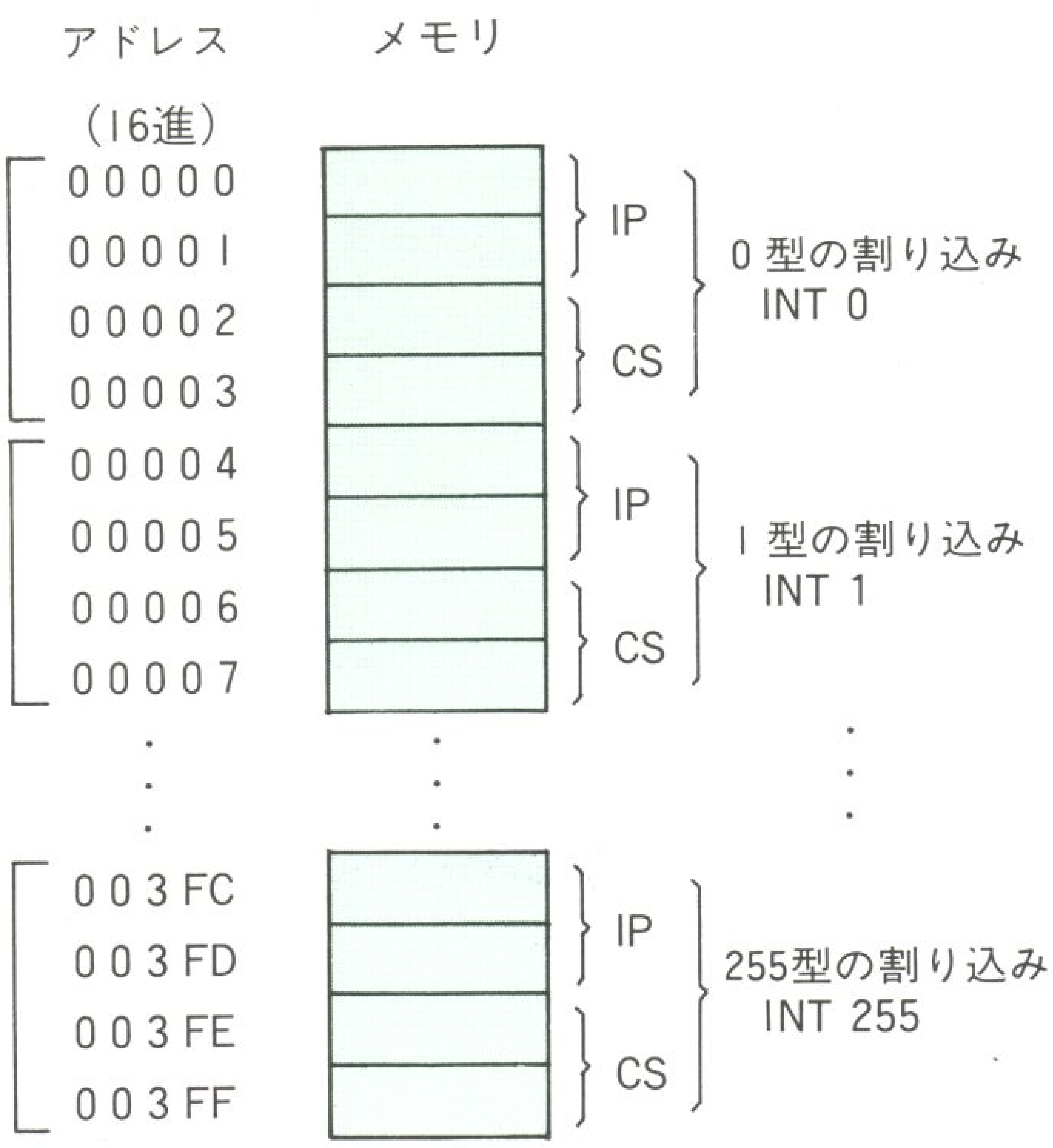
CPU も、この割り込みに対する処理を行うことができます。ユーザーがマシン語プログラム開発で利用できる割り込み処理を「ソフトウェア割り込み」といいます。アセンブリ言語では INT 命令（INTer-rupt：割り込む）を使います。

8086 では、物理アドレスの 00000H から 003FFH に、4 バイトずつを 1 つの要素として 256 個の要素が並べられて入っています。1 つの要素には、IP（命令ポインタ）と CS（コード・セグメント）が入っています。これは、ソフトウェア割り込みが起こったときに、処理される（実行される）プログラムの先頭アドレスです。

● 8086 は CS で示されるセグメント内で IP で示されるオフセットからプログラムの実行を始める。

1 つの要素を割り込みの「型」と呼び、0 型、1 型、. . . 255 型と呼びます。このアドレスの格納のされかたを図示すると、次のようになります。

■ソフトウェア割り込みのアドレスの格納のされ方



そして各型に応じた割り込み処理が、あらかじめプログラムされています。割り込みのアドレスと処理内容は、起動するハードウェアとソフトウェアのシステムによって異なります。同じ8086のパソコンでも、機種が違えば、異なりますし、同じ機種でもDISK BASIC, CP/M-86, MS-DOSをそれぞれ起動すると、各システムに応じたアドレスと処理内容に設定されます。が、処理内容は、システムのROM内のルーチン（プログラム）を呼び出し、ハードウェアを制御するものが一般的です。

ここで、MS-DOSでの割り込みアドレスを見てみましょう。SYMDEB（DEBUG）を起動して、次のように入力してください。

00000 ～ 003FF
4バイトずつ 256個の INTが
用意されている。

```
A>SYMDEB
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-D0:0,FF
0000:0000  63 4F 64 06 36 09 80 FD-F0 08 80 FD 36 09 80 FD  cOd.6..}p..}6..}
0000:0010  36 09 80 FD C8 52 60 00-F2 51 60 00 36 09 80 FD  6..}HR'.rQ'.6..}
0000:0020  59 1A 80 FD 44 0E 80 FD-37 09 80 FD 37 09 80 FD  Y..}D..}7..}7..}
0000:0030  DC 17 80 FD 37 09 80 FD-37 09 80 FD 37 09 80 FD  ¥..}7..}7..}7..}
0000:0040  37 09 80 FD 69 0C 00 D7-69 23 80 FD F7 22 80 FD  7..}i..Wi#.}w".}
0000:0050  37 09 80 FD 37 09 80 FD-37 09 80 FD 37 09 80 FD  7..}7..}7..}7..}
0000:0060  BD 0A 80 FD 96 15 80 FD-DB 5F 60 00 03 60 60 00  =..}...}[ '..''.
0000:0070  03 19 80 FD 36 09 80 FD-EC 27 80 FD 00 2A 00 D8  ...}6..}l'.}.*.X
0000:0080  60 11 64 06 81 11 64 06-28 09 33 11 C5 09 33 11  '.d...d.(.3.E.3.
0000:0090  F0 08 33 11 1D 13 64 06-4C 13 64 06 C8 5B 64 06  p.3...d.L.d.H[d.
0000:00A0  66 11 64 06 2C 5E 60 00-66 11 64 06 66 11 64 06  f.d.,^'.f.d.f.d.
0000:00B0  66 11 64 06 66 11 64 06-57 02 3B 10 59 14 64 06  f.d.f.d.W.;.Y.d.
0000:00C0  EA 67 11 64 06 11 64 06-66 11 64 06 66 11 64 06  jg.d..d.f.d.f.d.
0000:00D0  66 11 64 06 66 11 64 06-66 11 64 06 66 11 64 06  f.d.f.d.f.d.f.d.
0000:00E0  66 11 64 06 66 11 64 06-66 11 64 06 66 11 64 06  f.d.f.d.f.d.f.d.
0000:00F0  66 11 64 06 66 11 64 06-66 11 64 06 66 11 64 06  f.d.f.d.f.d.f.d.
-
```

INT 0　INT 1　INT 21H　INT 2　INT 3　INT20H　INT F0H　INT FFH

これは、セグメント 0. オフセット 0FFH の内容を表示したものです。はじめから 4 バイトずつの組になっていて、それぞれ順番に INT 0, INT 1, INT 2, INT 3, . . . INT FF が使用するアドレスが格納されています。

INT 0 を例にとるとアドレスの見方は、次のとおりです。

アドレス：	**0000**	**0001**	**0002**	**0003**
内　容：	**63**	**4F**	**64**	**06**
	（オフセット）		**（セグメント）**	

● 16 ビットの内容は、8 ビット 2 つ、すなわち 2 バイトにわけられる。このうち桁の高いバイト(上半分)を上位バイト、桁の低いバイト(下半分)を下位バイトという。

例:4F63H の場合
上位バイト:4FH
下位バイト:63H

メモリ上のデータが 2 バイトの場合、8086 では、アドレスは、その番号の高いほうから読むことになっています（セグメントは 0003, 0002, オフセットは 0001, 0000 と、それぞれ順に読む）。

セグメント・アドレスは、0664
オフセット・アドレスは、4F63

となります。

それで、INT 0が実行されると、セグメント・アドレス0664H、オフセット・アドレス4F63Hから格納されている割り込み処理プログラムが呼び出される（実行される）のです。

PC-9800シリーズでの割り込みの型とアドレス、その処理内容をまとめたのが、次の表です。

●メモリの内容は、アドレスの多いほうから少ないほうに読むが、これは逆のようであるが実は正しい。それは、16ビットの内容は、アドレスの低い順から下位バイト、上位バイトとメモリに格納されれため。上位、下位と読もうとすれば、当然アドレスの多いほうから読むことになる。

■ PC-9800シリーズのソフトウェア割り込み

INT	アドレス	内　容
5	14〜17	COPYキー　テキスト画面のハードコピーをとる。プリンタに出力する。
6	18〜1B	STOPキー　ディスクのリセット、カーソル・テキストの初期化などをする。
7	1C〜1F	割り込みタイマ IRET
8	20〜23	タイマ　｝ハードウェアによって割り込みをおこす。
9	24〜27	キーボード（キースキャン）｝ハードウェアによって割り込みをおこす。
18	60〜63	キーボード入力、画面制御
19	64〜67	RS-232C
1A	68〜6B	プリンタBIOS（カセット）
1B	6C〜6F	5インチ, 8インチフロッピーディスクBIOS
1C	70〜73	インターバルタイマ・カレンダ
1D	74〜77	IRET
20	80〜83	INT 20　MS-DOSファンクションコール・プログラム終了
21	84〜87	INT 21　MS-DOSファンクションコール
DC	370〜373	カーソル移動、ファンクションキー読み書き

●このソフトウェア割り込みの使い方については、「第3部3−3 INTコールあれこれ」(225ページ)を参照。

■ソフトウェア割り込みの実行

各型の割り込みを実行するには、アセンブリ言語では次のように書きます。

INT 型の番号

0型の割り込みを実行するには、次のようにします。

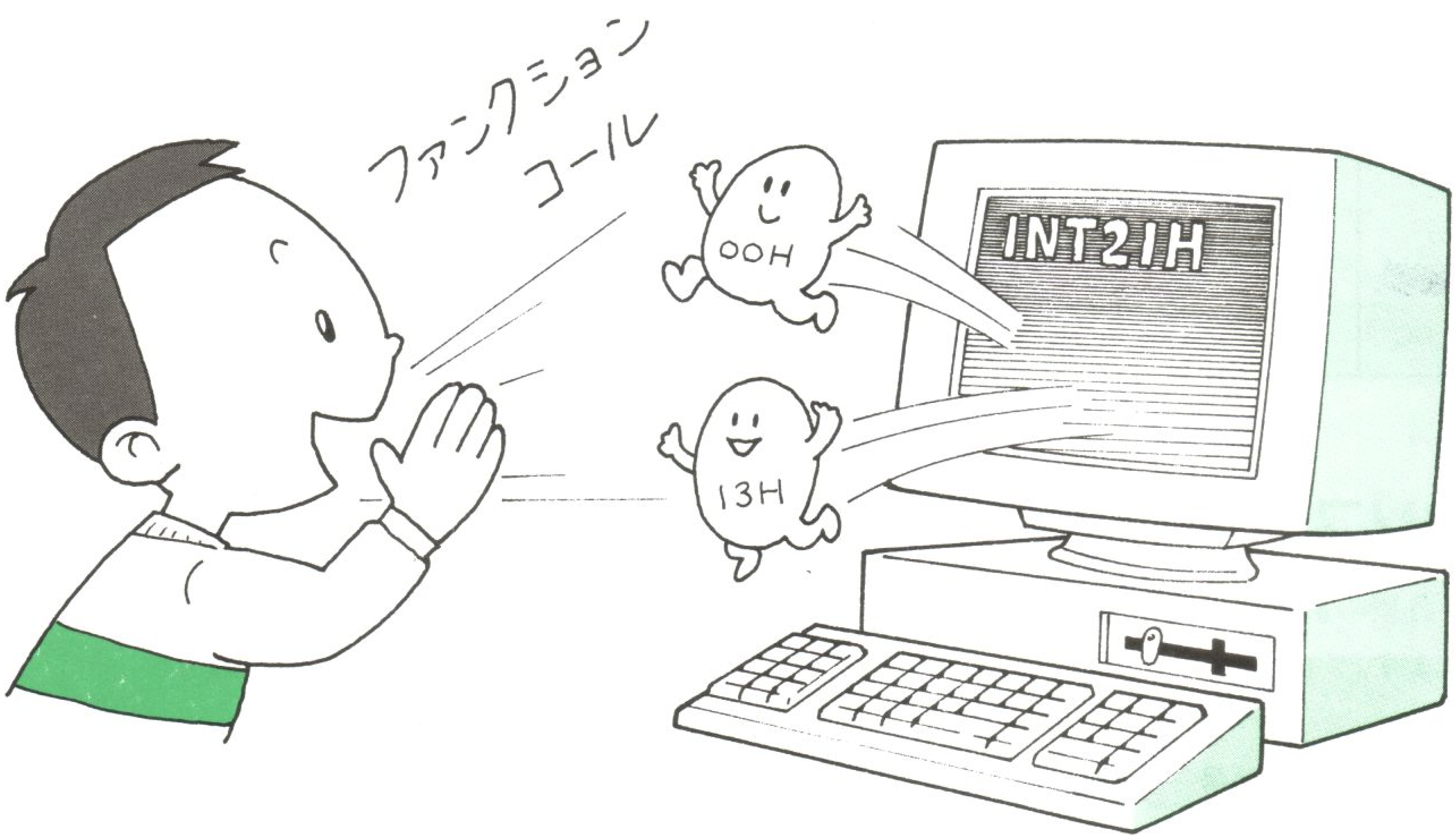
INT 0

これが実行されると、物理アドレス 00002, 00003 に格納されているセグメント・アドレスと 00001, 00002 に格納されているオフセット・アドレスに処理が移ります。

INT 21 H

は、16 進で 21 番の型の割り込みを実行することです。MS-DOS では、この INT 21H でユーザーがいろいろな機能を呼び出せるようになっています。これをファンクション・コール（Function Call : 機能呼び出し）といいます。

これまでも何度か INT 21H を実行してきましたが、MS-DOS のファンクション・コールの主な機能には、次頁の表のようなものがあります。



■ MS-DOS の主なファンクション・コール一覧

機能番号	機　　能	BASIC 相当
00H	プログラム終了	END
01H	キーボードから1文字入力	INPUT$（1）
02H	画面に1文字表示	PRINT "A"
05H	プリンタへの1文字出力	LPRNT "A"
09H	画面に文字列出力	PRINT "STRING"
0AH	キーボードから1行入力	INPUT A$
0FH	ファイルオープン	OPEN
10H	ファイルクローズ	CLOSE
11H	ファイル名サーチ	——
12H	次のファイル名サーチ	——
13H	ファイル削除	KILL
14H	シーケンシャルリード	INPUT # 1, A$
15H	シーケンシャルライト	PRINT # 1, A$
16H	新規ファイルクリエイト	OPEN
17H	ファイル名変更	RENAME
1AH	バッファ（DTA）アドレスのセット	LSET/RSET
21H	ランダムリード	GET # 1, R
22H	ランダムライト	PUT # 1, R
24H	ランダムレコード番号のセット	——
3CH	新規ファイル作成	OPEN
3DH	オープンファイル	OPEN
3EH	クローズファイル	CLOSE
3FH	ファイルリード	INPUT # 1, A$
40H	ファイルライト	PRINT # 1, A$
41H	ファイル削除	KILL
4CH	プロセスの終了	END
4EH	ファイル名サーチ	——
4FH	次のファイル名サーチ	——

●代表的なファンクション・コールの使い方は、「第2部3章アセンブリ言語をさらに学ぶ(107ページ)と「第3部3章独自の実用コマンドを開発する」(191ページ)を参照。

基本的な使い方は、レジスタ AH に、機能の番号を入れて、INT 21H を実行します。

例：キーボードから1文字入力する

```
MOV AH, 1
INT 21 H
```

　また、実行するときDXなど他のレジスタにも値を入れる必要があるものもあります。

1·5 レジスタの機能と役割

8086のレジスタは、機能別に、次の3つのグループと2つの特殊レジスタに分類できることは、すでに述べました。

1. データ処理レジスタ・グループ
2. アドレス指定レジスタ・グループ
3. セグメント指定レジスタ・グループ
4. 命令ポインタ・レジスタ
5. フラグ・レジスタ

アセンブリ言語でプログラムを書くには、これらのレジスタの機能と役割を理解しておく必要があります。以下、特に覚えておきたいレジスタについて詳しく説明します。

1. データ処理レジスタ・グループ

データの四則演算、掛算、割り算、論理演算を主に行うレジスタで、次の4つがあります。

AX：Accumulator（アキュムレータ）
BX：Base Register（ベース・レジスタ）
CX：Count Register（カウント・レジスタ）
DX：Data Register（データ・レジスタ）

<AX（アキュムレータ）>

アキュムレータとは、「蓄積するもの」という意味で、加算や乗算のときなどに答えをためて蓄積することが多いので、こう呼ばれます。AXは16ビットのレジスタですが、8ビットにも対応できるようになっており、2つの8ビット・レジスタにわけて使うことができます。

AX →AH（AXのHigh Register）
**　　AL（AXのLow Register）**

Highというのは、16ビットの上半分で、Lowは下半分を表わします。例えば、AXに16進で1234という値が入っていれば、

●AHには、AX16ビットの上半分(上位バイト)が入り、ALには下半分(下位バイト)が入る。

● AX＝1234H のとき、次のようにわけられる。

上位バイト AH＝12H
下位バイト AL＝34H

● AH＝56H, AL＝78H のとき、次のようにくっつけられる。

AX＝5678H

● AH には、MS-DOS のファンクション・コールで機能番号を入れる。

AH＝12

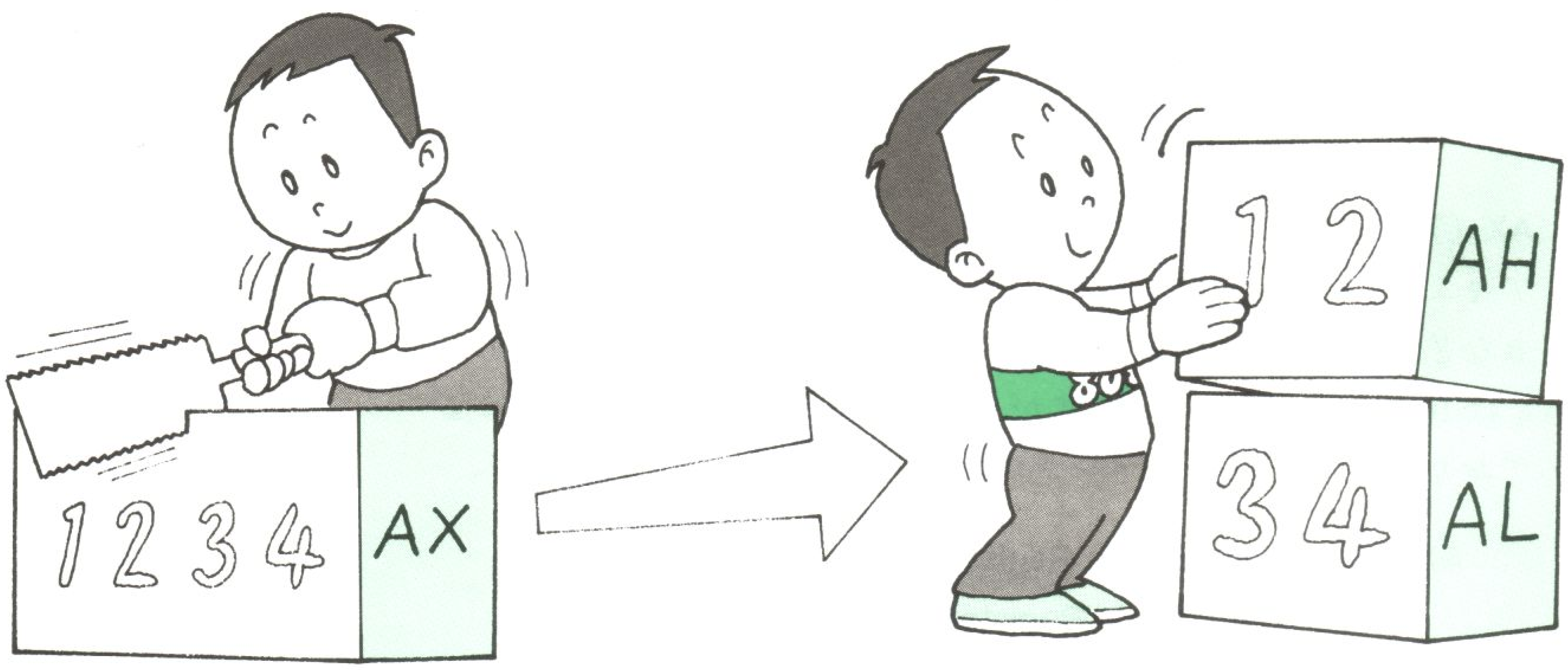
AL＝34

とわけられます。これは、何も計算してわけるのではなく、ただ単純にわけて使うことができます。

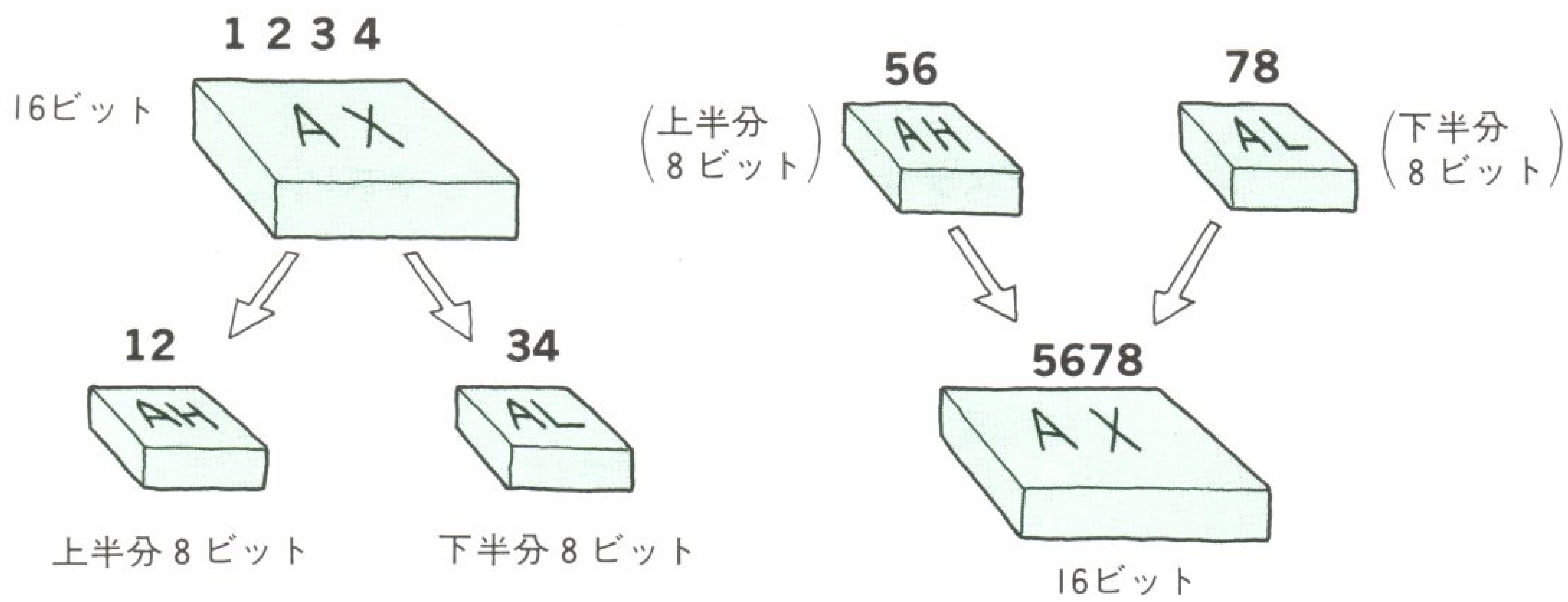
逆に、AH＝56, AL＝78 と入っていれば、

AX＝5678

として使うことができます。



■AXレジスタの分割と統合



<BX（ベース・レジスタ）>

主に、メモリをアクセスするときに、アドレスをBXに入れ、そこをベース(基準)として、メモリ内容の読み書きに用います。BXもBHとBLとに分割できます。

● BXは、アドレス指定に使う。

<CX（カウント・レジスタ）>

主に、ループ回数を指定するのに用い、回数のカウントができます。CXもCHとCLとに分割できます。

● CXは、カウンタとして使う。

<DX（データ・レジスタ）>

主に、乗算や除算に使います。DXもDHとDLとに分割できます。

● DXには、MS-DOSのファンクション・コールでデータが格納されているアドレスを入れる。また、DLには1文字表示やプリンタ出力のアスキーコードを入れる。

2. アドレス指定レジスタ・グループ

レジスタのデータをメモリに格納したり、メモリからレジスタにデータを入れるときに、アドレスを指定する必要がありますが、このアドレスを入れるレジスタです。次の4つがあります。

SI：Source Index（ソース・インデックス）
DI：Destination Index（デスティネーション・インデックス）
SP：Stack Pointer（スタック・ポインタ）
BP：Base Pointer（ベース・ポインタ）

● SI,DI はアドレス指定に使う。

<SI（ソース・インデックス）>

アドレス指定に用い、特にデータ転送のときの転送元（ソース）のアドレスを指定します。

<DI（デスティネーション・インデックス）>

アドレス指定に用い、特にデータ転送のときの転送先（デスティネーション）のアドレスを指定します。

SP と BP は、入門の段階では、使いません。

3. セグメント指定レジスタ・グループ

アドレスを指定する前に、どのセグメントを選ぶかという大前提があります。このセグメントを入れるレジスタです。次の4つがあります。

CS : Code Segment（コード・セグメント）
DS : Data Segment（データ・セグメント）
ES : Extra Segment（エキストラ・セグメント）
SS : Stack Segment（スタック・セグメント）

● MS-DOS のマシン語ファイルでは、CS の値は自動的に設定される。

<CS（コード・セグメント）>

コードすなわちプログラムそのものを格納する領域を指定します。8086 では、CS で示される領域内の IP : Instruction Pointer（インストラクション・ポインタ）が示すアドレスが実行中のアドレスとなります。MS-DOS では、自動的に CS の値が設定され、ユーザーは設定しません。

● MS-DOS のマシン語ファイルでは、DS の値は自動的に設定される。また、別のアドレスに設定することもある。

<DS（データ・セグメント）>

データを格納する領域を指定します。8086 の命令がデータをアクセスするときは、通常このデータ・セグメントの領域に行います。この中のアドレスを示すのに BX, SI, DI を使います。

● COM ファイルでは、CS=DS=ES とすることが多いが、ESはDSと別のアドレスに設定することもある。

<ES（エキストラ・セグメント）>

DS で指定するデータ領域が足りなくなったり、別な領域にもデータ領域が必要なときに使います。補助データ領域（エキストラ）の指定をします。この中のアドレスを示すのにも BX, SI, DI を使います。

<SS（スタック・セグメント）>

入門の段階では、使いませんし、普通 MS-DOS が指定します。

4. 命令ポインタ・レジスタ（IP インストラクション・ポインタ）

命令を実行するアドレスを指し示すもので、プログラム領域で実行する命令のアドレスが自動的に入ります。ユーザーは、指定しません。

●IP は、COM ファイルでは必ず 100H と設定される。

5. フラグ・レジスタ

演算結果や状態を示すレジスタです。処理に応じて自動的に結果や内容が入ります。フラグは、0, 1 で表わすため、フラグ・レジスタの各ビットを 9 つの機能を持つフラグにしています。ここでは、そのなかで次の 4 つのものがあることを知っておいてください。

●フラグは、条件判断に使われる。

< OF（Overflow Flag）>

オーバーフローフラグで、計算の結果、桁が大きすぎて処理しきれなかったことを示します。

< SF（Sign Flag）>

サインフラグで、計算の結果がマイナスになったことを示します。

< ZF（Zero Flag）>

ゼロフラグで、計算の結果が 0 になったことを示します。

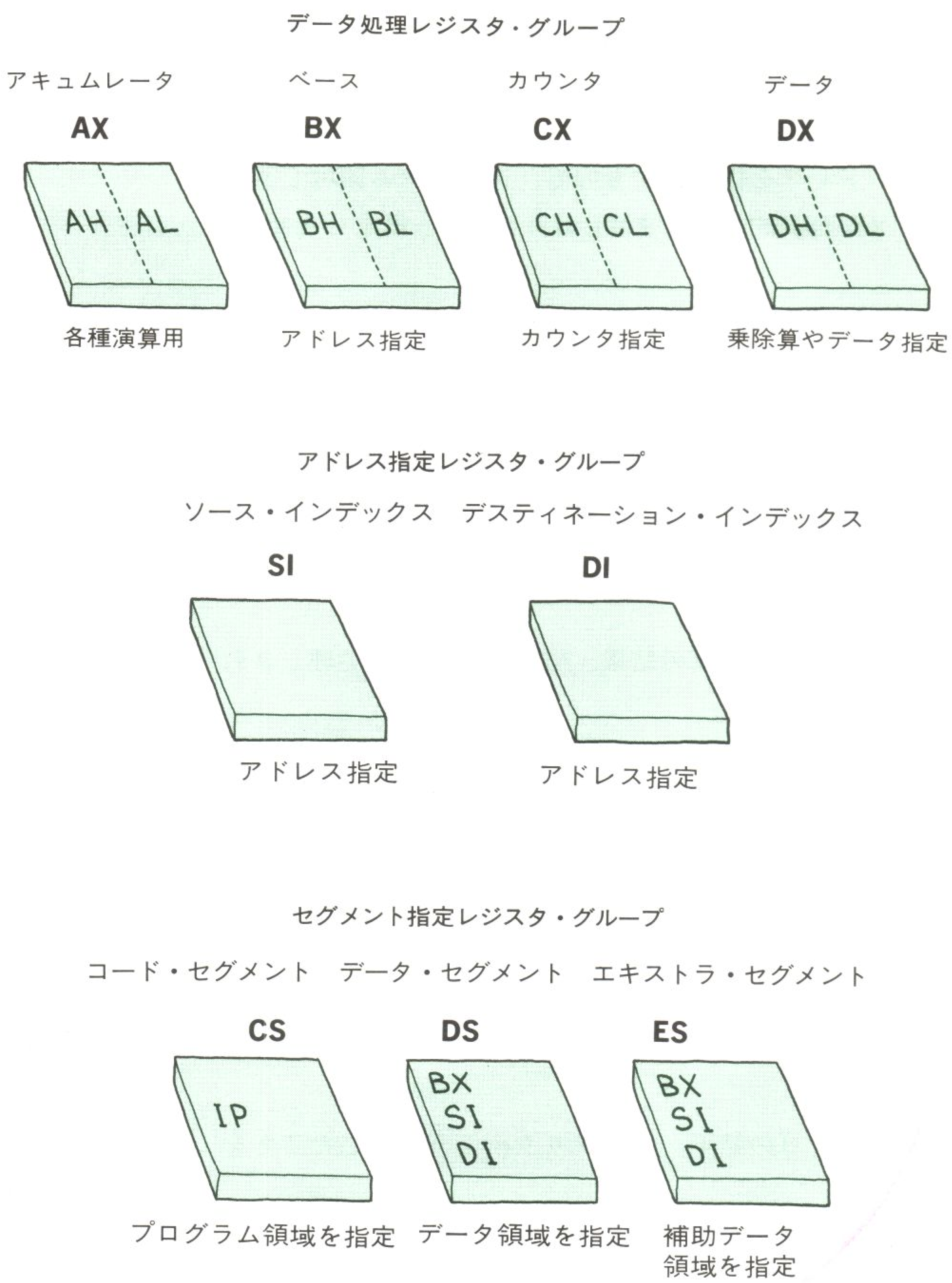
< CF（Carry Flag）>

キャリーフラグで、計算の結果、桁上がりや桁借りが生じたことを示します。

このように、フラグは、各レジスタを使って計算や演算などを行ったときに、その結果を判定することができます。BASIC 言語では、IF 文がありますが、アセンブリ言語ではこれがないので、フラグで判定することになります。

8086 のレジスタとその役割をまとめて図にすると、次のようになります。

■レジスタの機能一覧（よく使うもの）



これらのレジスタの機能は、これから実際にプログラミングするうちに明確にわかってきます。ここでは、予備知識としておいてください。

第2部

第2章 マクロアセンブラについての知識

ここでは、第1部第3章で取り上げたアセンブリ言語のソース・プログラムYES NO.ASMを例にして、説明をします。これまでアセンブラに関して簡単に説明してきたことをまとめて復習する意味を含めてさらに詳しく解説することにします。

2・1 アセンブラとは

すでに皆さんは、マシン語がどういうものかがおわかりでしょう。英数字 (16進数) で表現されているマシン語は、我々には (プロのプログラマでさえ)、とうてい理解できるものではありません。さらに、マシン語でプログラミングすることは不可能といえます。あの意味もない、16進表現のマシン語のら列では、第一覚えようがないですし、万一覚えられたとしても、間違いが生じるのは火をみるより明らかです。

そこで、我々がプログラミングできるように、マシン語の各命令に対応する記号が考案されました。この記号を**ニーモニック** (Mnemonic：記憶を助けるもの) といいます。ニーモニックは、マシン語1つ1つに付けられたニックネームと考えてください。

●アセンブリ言語の表記は、ニーモニックで行う。ニーモニックは、マシン語の命令に付けられたニックネーム。

●この例は8086のマシン語とニーモニック。

●MOVは、値を代入するマシン語命令のニーモック。

例：レジスタ AL に 1 を入れる

マシン語	ニーモニック
B001	MOV AL, 1 ――― アセンブリ言語

B001 では、なんのことだかわからないが、
MOV とは MOVe の略で、移すことと意味がわかる。

これらのニーモニックをいろいろ組み合わせて、プログラムを表現するものがアセンブリ言語です。アセンブリ言語は、BASIC言語ほどではないにしても英語の単語やその省略形を使っており、マシン語に比べると格段にわかりやすいものです。そして、アセンブリ言語は、マシン語の命令と1対1に対応していますので、マシン語を開発することは、アセンブリ言語でプログラムを書くことに等しいのです。

アセンブリ言語で書いたプログラムをマシン語のプログラムに直すプログラムが**アセンブラ**です。アセンブリ言語とマシン語の命令とは、1対1に対応していますので、アセンブラは、基本的には単純にアセンブリ言語をマシン語に変換することになります。なお、アセンブリ言語をマシン語に変換することを**アセンブル**するといいます。

●マシン語とアセンブリ言語は、CPUによって異なる。

■マシン語とアセンブリ言語の関係（8086の場合）

マシン語 ← アセンブリ言語

アセンブル

1対1に対応

```
Microsoft MACRO Assembler  Version 3.00               Page    1-1
                                                              07-23-87

                                    ;-------------------------
                                    ;   YESNO.ASM
                                    ; YES OR NO Input
                                    ; for Batch file
                                    ;-------------------------
         0000                       CODE SEGMENT
                                         ASSUME CS:CODE,DS:CODE
         0100                            ORG 100H
                                    ;---------------------------------
         0100                       START:
         0100  B4 01                     MOV AH,1        ; AL$=INPUT$(1)
         0102  CD 21                     INT 21H         ; PRINT AL$
                                    ;---------------------------------
         0104  3C 59                     CMP AL,'Y'      ; IF AL$="Y" THEN
         0106  74 0E                     JE YES          ; GOTO YES
         0108  3C 79                     CMP AL,'y'      ; IF AL$="y" THEN
         010A  74 0A                     JE YES          ; GOTO YES
                                    ;---------------------------------
```

アセンブラとコンパイラとの違い

BASIC言語やC言語、COBOL、FORTRANなどの高級言語で書いたプログラムは、コンパイラと呼ばれるプログラムでマシン語に直されますが、この場合、高級言語の１つの命令とそれに対応してマシン語になった命令は、１対１に対応していません。

通常は、高級言語の１つの命令に対して、複数のマシン語の命令になります。この点が、アセンブラとコンパイラの違いです。

●アセンブラは、ソース・プログラムの命令と１対１に対応したマシン語を作るが、コンパイラは、ソース・プログラムの１命令につき複数のマシン語を作る。

2・2 マクロアセンブラとは

マクロとは英語の「macro」からきた言葉で「大きい」、「長い」という意味です。これは、「小さい」、「微小」という意味のミクロ「micro」に対応するものです。

コンピュータ用語では、細かい基本的な個々の命令を一般にミクロ命令といい、ミクロ命令をいくつか組み合わせて、あるまとまった処理をするものをマクロ命令といいます。

アセンブリ言語でプログラムを書くにあたって、ある1つの処理をするのに、いくつかの命令を組み合わせて書きますが、このとき、これらの命令をまとめて1つの命令にすることができれば、プログラムの開発効率が上がるとともに間違いも少なくなります。これをマクロ命令といいます。

●マクロ命令とは、複数の命令を1つにまとめて実行できるもの。

マクロ命令を使うと、ある処理をするときに個々の命令を順次書く必要がなくなり、1つのマクロ命令で代用することができます。このマクロ命令が使えるアセンブラがマクロアセンブラなのです。

マクロアセンブラの特長

アセンブリ言語でプログラムを開発するとき、マクロアセンブラを使うと、次のようなことができマシン語開発の効率が上がります。

1. ラベルが使える
2. コメントが書ける
3. 数値やアドレスに名前が付けられる
4. エラーを表示する
5. 数値は10進、16進などが使える
6. 豊富な疑似命令が使える
7. 各種演算子が使える
8. マクロ命令が使える
9. 各種ファイルが利用できる
10. 高級言語とリンクできる

以下、各特長について説明します。

●ラベルの先頭は必ず英字にしなければならない。長いラベルにするときは、_(下線)でつなぐことができる。

例:PROGRAM_END

1. ラベルが使える

分岐先、サブルーチン、メモリの各アドレスを指定するのにラベル

が使え、実際のアドレスを考える必要がなく、わかりやすいプログラムが書けます。

例：**JMP START** 　　**CALL DISPLAY**

次の　　　　部分が分岐先の位置と分岐命令にラベルを使った例です。

■ラベルの使用例

```
;-------------------------
;    YESNO.ASM
; YES OR NO Input
; for Batch file
;-------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
;---------------------------------
START:
     MOV AH,1        ; AL$=INPUT$(1)
     INT 21H         ; PRINT AL$
;----------------------------------
     CMP AL,'Y'      ; IF AL$="Y" THEN
     JE YES          ; GOTO YES
     CMP AL,'y'      ; IF AL$="y" THEN
     JE YES          ; GOTO YES
;----------------------------------
     CMP AL,"N"      ; IF AL$="N" THEN
     JE NO           ; GOTO NO
     CMP AL,"n"      ; IF AL$="n" THEN
     JE NO           ; GOTO NO
;----------------------------------
     JMP START       ; GOTO START
;----------------------------------
YES:
     MOV AL,0        ; AL=0
     JMP BACK        ; GOTO BACK
;----------------------------------
NO:
     MOV AL,1         ; AL=1
BACK:
     MOV AH,4CH       ; Return AL(0,1)
     INT 21H          ; to Batch file
;----------------------------------
CODE ENDS
     END START
```

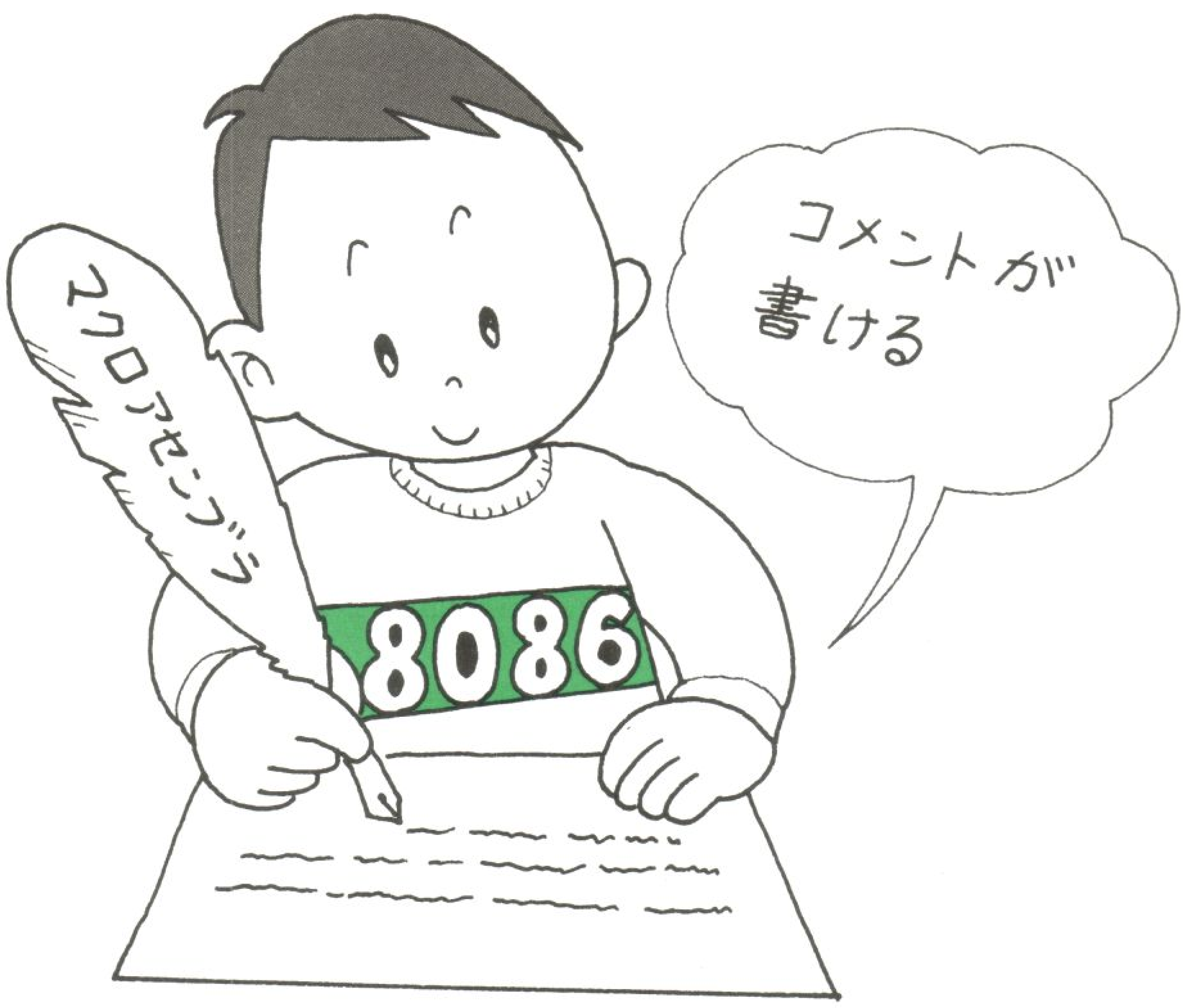
分岐命令で飛び先にラベルを使っている
この場合、ラベルのあとに必ず：(コロン) を付ける

ラベル分岐先の位置

2. コメントが書ける

BASIC言語のREMや「'」のように、「；」を先頭に付けると、それ以降はアセンブルの際に何も影響を与えないコメント行となります。コメントを書くことにより、プログラムを見やすく分かりやすくすることができます。

```
例：　;---------------
      ; Display Sub
      ;---------------
```



次の　　　　部分がコメントです。

■コメントの使用例

```
;-------------------------
;    YESNO.ASM
; YES OR NO Input
; for Batch file
;-------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
;-----------------------------------
START:
     MOV AH,1        ; AL$=INPUT$(1)
     INT 21H         ; PRINT AL$
;-----------------------------------
     CMP AL,'Y'      ; IF AL$="Y" THEN
     JE YES          ; GOTO YES
     CMP AL,'y'      ; IF AL$="y" THEN
     JE YES          ; GOTO YES
;-----------------------------------
     CMP AL,"N"      ; IF AL$="N" THEN
     JE NO           ; GOTO NO
     CMP AL,"n"      ; IF AL$="n" THEN
     JE NO           ; GOTO NO
;-----------------------------------
     JMP START       ; GOTO START
;-----------------------------------
YES:
     MOV AL,0        ; AL=0
     JMP BACK        ; GOTO BACK
;-----------------------------------
NO:
     MOV AL,1         ; AL=1
BACK:
     MOV AH,4CH       ; Return AL(0,1)
     INT 21H          ; to Batch file
;-----------------------------------
CODE ENDS
     END START
```

コメント

コメント

;以降はコメント
;------ 線を引いて区分しプログラムを見やすくもできる

3. 数値やアドレスに名前が付けられる

よく使う数値や参照するアドレスに名前が付けられます。これで、名前で参照することができます（6の疑似命令参照）。

```
例： CR EQU 13
     LF EQU 10
     VRAM EQU 0A000H

    BASIC 言語では、次のとおり。

    CR$＝CHR$(13)
    LF$＝CHR$(10)
    VRAM＝&HA000
```

4. エラーを表示する

マクロアセンブラがソースプログラムをアセンブルするとき、エラーがあれば、その箇所とエラーの内容を表示します。プログラムの誤りをすぐ見つけることができます。

```
例：Error－－－10：syntax error
```

5. 数値は10進，16進などが使える

数値は10進はもとより16進、2進、8進の数が使え、必要に応じて使い分けることができます。

```
例： 10進で16の数を表わす場合
     10進  16（なにもつけずにそのままの数）
     16進  0FH（Hexadecimal：16進のH）
      2進  00001111B（Binary：2進のB）
      8進  10Q（Octadecimal：8進のOの代りにQ）
```

普通は10進と16進を使います。

6. 豊富な疑似命令が使える

8086の命令ではなく、アセンブラに対する命令を疑似命令といいます。これは、アセンブラがソースプログラムをアセンブルするときに必要な情報を与えるものです（2－3の疑似命令参照）。

例：VRAM EQU 0A000H
MOV AX, VRAM
（AX に 0A000H が入る）
EQU とは、EQUal（＝）の略です。

7. 各種演算子が使える

＋－＊／の四則演算子と AND, OR などの論理演算子が使え、数値をいちいち計算する必要がなく、かつわかりやすくもなります。

例：MOV CX, 80 ＊ 25
（CX に 80 ＊ 25 の値を入れる。）
MOV DX, OFFSET CRLF＋1
（DX に CRLF のアドレスに＋1 したものを入れる。）

8. マクロ命令が使える

いくつかのアセンブリ言語の命令を1つの命令（マクロ命令）にして使うことができます。

●マクロ命令を使うと確かに便利だが、入門ではわかりづらいので本書では、使っていません。熟練したらどんどん使うとよい。

例：MS-DOS のファンクションコールのマクロ定義

```
FCALL MACRO FNO
        MOV AH, FNO
        INT 21H
        ENDM
```

使い方は、

```
FCALL 1
```

とすると

```
MOV AH, 1
INT 21H
```

と同じことになり、1文字入力のファンクションコールとなります。
また、

```
FCALL 2
```

とすると

```
MOV AH, 2
INT 21H
```

と同じことになり、1文字出力のファンクションコールとなります。

9. 各種ファイルが利用できる

1つのプログラムを開発するのに、次のファイルができます(XXXはソースファイル名)。

XXX. ASM（ソース・プログラム）
XXX. OBJ（オブジェクトファイル）
XXX. LST（リストファイル）
XXX. EXE（EXE ファイル）
XXX. COM（COM ファイル）
XXX. CRF（クロスリファレンス・ファイル）

●リストファイルは、行番号付きでソース・プログラムとアセンブルされたマシン語を対比した内容をもつファイル。また、使われているラベル(シンボル)一覧もわかる。

オブジェクトファイルは、10のように高級言語のオブジェクトファイルとリンクできますし、リストファイルは、ソース・プログラムとマシン語との対比がわかります。またクロスリファレンス・ファイルはCREFコマンドによって、ラベルと実際のアドレスの相互参照（Cross Reference：クロスリファレンス）ができます。

●クロスリファレンス・ファイルは、ラベルがどの行で設定され、ラベル名と実際のアドレスの対比がどうなっているかを記録したファイル。この内容をみるには、CREF コマンドでテキスト・ファイルにする必要がある。

■アセンブルのときの各種ファイル作成

```
A>MASM YESNO ⏎
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

Object filename [YESNO.OBJ]: ⏎
Source listing  [NUL.LST]: YESNO ⏎
Cross reference [NUL.CRF]: YESNO ⏎

49698 Bytes free

Warning Severe
Errors  Errors
0       0

A>
```


■リストファイルとクロスリファレンス・ファイルが作成されている

```
A>DIR YESNO.* ⏎

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

YESNO    EXE      801  87-07-22  16:49
YESNO    COM       33  87-07-22  16:51
YESNO    ASM      984  87-07-22  17:24
YESNO    OBJ       76  87-07-23   0:29
YESNO    LST     2233  87-07-23   0:29
YESNO    CRF      110  87-07-23   0:29
        6 個のファイルがあります.
    34816 バイトが使用可能です.

A>
```

■リストファイル

```
    Microsoft MACRO Assemble-  Version 3.00                 Page     1-1
                                                                     07-23-87
 行番号        マシン語                              ソース・プログラム
   1                                      ;-------------------------
   2                                      ;   YESNO.ASM
   3                                      ; YES OR NO Input
   4                                      ; for Batch file
   5                                      ;-------------------------
   6       0000                           CODE SEGMENT
   7                                           ASSUME CS:CODE,DS:CODE
   8       0100                                ORG 100H
   9                                      ;---------------------------------
  10       0100                           START:
  11       0100    B4 01                       MOV AH,1      ; AL$=INPUT$(1)
  12       0102    CD 21                       INT 21H       ; PRINT AL$
  13                                      ;----------------------------------
  14       0104    3C 59                       CMP AL,'Y'    ; IF AL$="Y" THEN
  15       0106    74 0E                       JE YES        ; GOTO YES
  16       0108    3C 79                       CMP AL,'y'    ; IF AL$="y" THEN
  17       010A    74 0A                       JE YES        ; GOTO YES
  18                                      ;----------------------------------
  19       010C    3C 4E                       CMP AL,"N"    ; IF AL$="N" THEN
  20       010E    74 0B                       JE NO         ; GOTO NO
  21       0110    3C 6E                       CMP AL,"n"    ; IF AL$="n" THEN
  22       0112    74 07                       JE NO         ; GOTO NO
  23                                      ;----------------------------------
  24       0114    EB EA                       JMP START     ; GOTO START
  25                                      ;----------------------------------
  26       0116                           YES:
  27       0116    B0 00                       MOV AL,0      ; AL=0
  28       0118    EB 03 90                    JMP BACK      ; GOTO BACK
  29                                      ;----------------------------------
  30       011B                           NO:
  31       011B    B0 01                       MOV AL,1       ; AL=1
  32       011D                           BACK:
  33       011D    B4 4C                       MOV AH,4CH     ; Return AL(0,1)
  34       011F    CD 21                       INT 21H        ; to Batch file
  35                                      ;----------------------------------
  36       0121                           CODE ENDS
  37                                           END START
```

■リストファイルの続き

```
 Microsoft MACRO Assembler  Version 3.00                Page    Symbols
                                    -1
                                                                07-23-87

Segments and Groups:

                N a m e                     Size    Align   Combine Class

CODE . . . . . . . . . . . . . . .0121      PARA    NONE

Symbols:  ────────────────────── 使用されているラベル(シンボル)一覧

                N a m e                     Type    Value   Attr

BACK . . . . . . . . . . . . . . .L NEAR    011D    CODE
NO . . . . . . . . . . . . . . . .L NEAR    011B    CODE
START. . . . . . . . . . . . . . .L NEAR    0100    CODE
YES. . . . . . . . . . . . . . . .L NEAR    0116    CODE

49698 Bytes free

Warning Severe
Errors  Errors
0       0
```

■クロスリファレンス・ファイルをクロスリファレンス・リスト・ファイルにする

```
A>CREF YESNO ⏎
Microsoft Cross Reference Utility  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

List filename [YESNO.REF]:⏎

A>DIR YESNO.*⏎

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥
                               ── クロスリファレンス・リスト・ファイルが作られている
YESNO     EXE      801  87-07-22  16:49
YESNO     COM       33  87-07-22  16:51
YESNO     REF      405  87-07-23   0:34
YESNO     ASM      984  87-07-22  17:24
YESNO     OBJ       76  87-07-23   0:29
YESNO     LST     2233  87-07-23   0:29
YESNO     CRF      110  87-07-23   0:29
         7 個のファイルがあります.
     33792 バイトが使用可能です.

A>
```

■クロスリファレンス・リスト・ファイルの内容

```
A>TYPE YESNO.REF ⏎
```

ラベル(シンボル)　参照行番号　#がついている行でラベルが定義されている

```
 Symbol Cross Reference                         (# is definition)      Cref-1

BACK . . . . . . . . . . . . . . .    28     32#

CODE . . . . . . . . . . . . . . .     6#     7      7      36

NO . . . . . . . . . . . . . . . .    20     22     30#

START. . . . . . . . . . . . . . .    10#    24     37

YES. . . . . . . . . . . . . . . .    15     17     26#

5 Symbols

62958 Bytes Free

A>
```

なお、ラベルや変数・アドレスに付けた名前を総称して、シンボルといいます。

10. 高級言語とリンクできる

C言語, PASCAL, COBOL, FORTRANなどのコンパイラは、MASMの出すオブジェクトファイルと同じマシン語ファイルを生成することができますので、LINKによりアセンブラで作ったマシン語とつなぎあわせることができます。

2·3 マクロアセンブラの命令

マクロアセンブラでソース・プログラムをアセンブルするためには、次の2種類の命令を使わなければなりません。

1. マシン語の命令に対応した命令（アセンブリ言語）
2. アセンブラに対する命令（疑似命令）

■マシン語の命令に対応した命令（アセンブリ言語）

8086の命令は、機能別に大きく分けて次の6つのグループにまとめられます。

●代表的なものは、MOV。

1. データ転送命令
レジスタ同士、レジスタとメモリとの間でのデータやり取り。

●代表的なものは、ADD, SUB,INC,DEC。

2. 算術演算命令
足し算、引き算、掛算、割り算を行う。

3. 論理演算命令
AND, OR などの論理演算を行う。

●代表的なものは、MOVSB, MOVSW。

4. ストリング（一続きのメモリ）操作命令
データの一括転送、サーチ、比較といったひと続きのメモリを取り扱う処理を行う。

●代表的なものは、JMP, CALL。

5. 分岐命令
ジャンプやサブルーチンコールなど処理の流れを変える。

6. CPU制御命令
主に8086のフラグの制御を行う。

これらの命令は、本書では必要に応じて1つのプログラムの中で、使い方を解説していますので、ここでは個々の命令の使い方の説明は割愛します。命令は、組み合わせて使い、覚えるほうがよいからです。

■アセンブラに対する命令（疑似命令）

ソース・プログラムをアセンブルするときに、アセンブラに対して使う命令です。これらを使いこなさないとアセンブルできませんし、使いこなすと、高級言語に近いプログラミングができます。

特に覚えておきたい疑似命令は、次のとおりです。

SEGMENT（セグメント）

セグメントの初めを指定

ENDS（エンドセグメント：END Segment）

セグメントの終わりを指定

ASSUME（アシューム：仮定するの意）

セグメント・レジスタを設定

EQU（イクオール：EQUal）

数値をラベルに定義、再定義不可

＝（イクオール）

数値をラベルに定義、再定義可

ORG（オリジン：ORiGin）

開始アドレス（オフセット）の設定

OFFSET（オフセット）

メモリ・アドレスを指定

DB（デファインバイト：Define Byte）

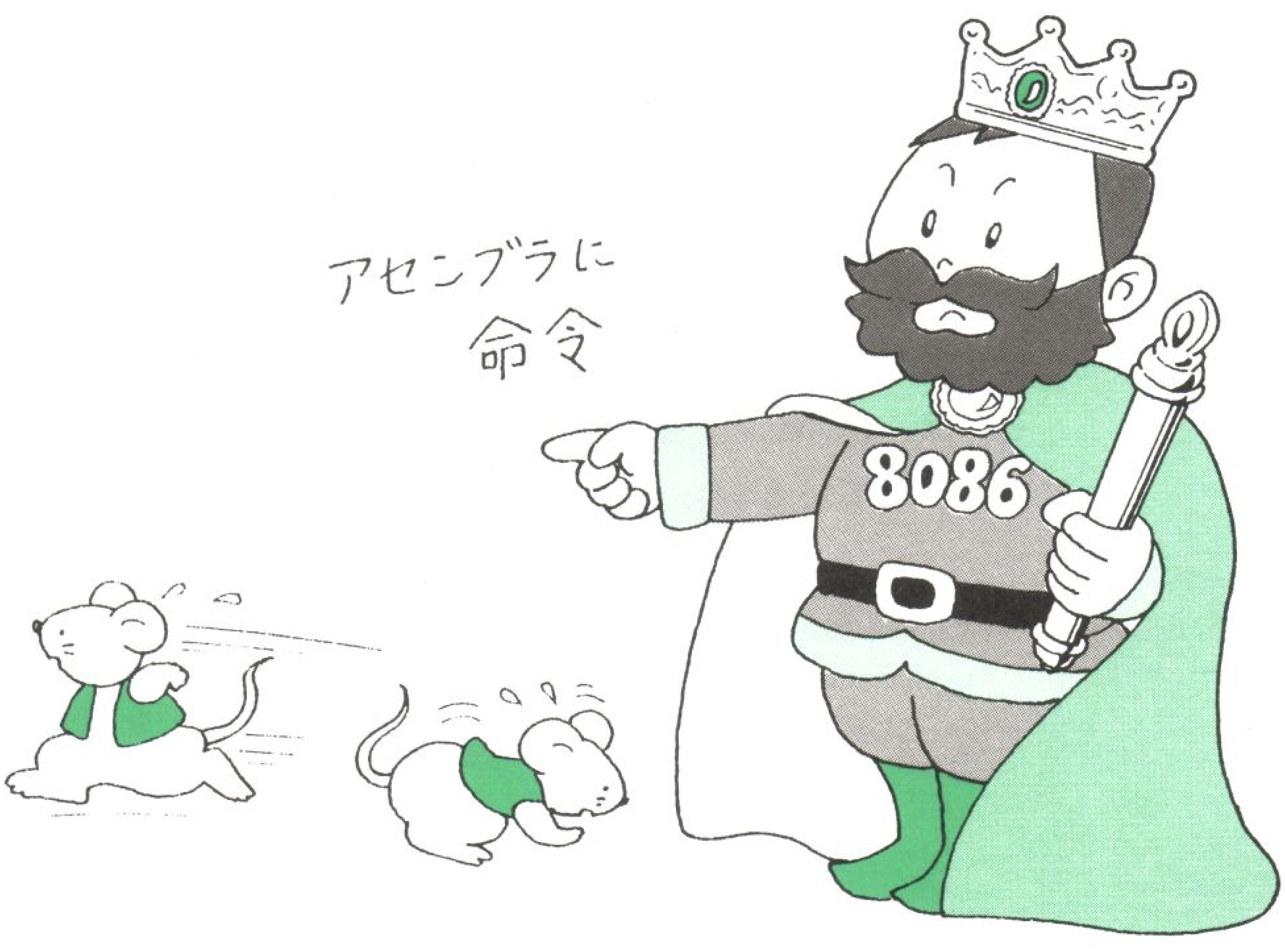
バイト単位で、データをメモリに格納

END（ソースプログラムの終わり）

例 1：COM ファイル専用のソース・プログラムの書き方

```
CODE SEGMENT
        ASSUME CS：CODE：DS：CODE
        ORG 100H
START：
            ⋮
プログラムそのもの
            ⋮
CODE ENDS
        END START
```

- はじめの 1 行は、“CODE SEGMENT”～“CODE ENDS”の部分を、CODE という名前のセグメントである、と宣言している。次の 1 行で、CS と DS の値を CODE の指すアドレスと同じに設定する。
- ORG で開始アドレスをオフセット 100H からに設定する。
- START でプログラムの開始位置を設定する。
- プログラムそのものを書く。
- CODE ENDS で CODE というセグメントを終わりにする。
- END START で START から始まるプログラムの終わりを設定する。



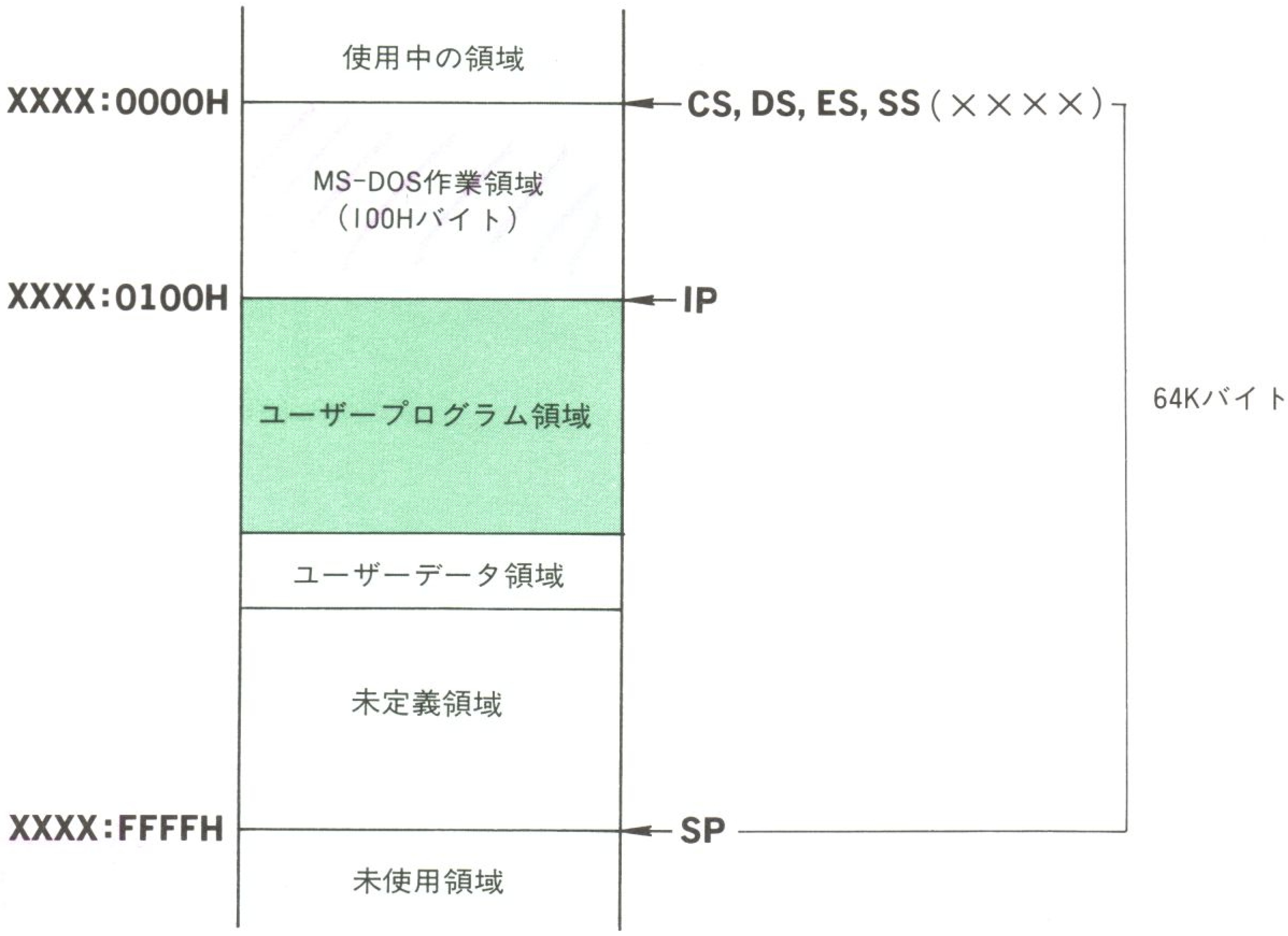
■COMファイル専用のソース・プログラムの書き方

```
;--------------------------
;    YESNO.ASM
; YES OR NO Input
; for Batch file
;--------------------------
CODE SEGMENT                          ── CODEの部分をセグメントとする
     ASSUME CS:CODE,DS:CODE           ── CSとDSをCODEのセグメントに設定する
     ORG 100H                         ── 開始アドレスを100Hとする
;----------------------------------
START:                                ── プログラムそのものの初めを設定
     MOV AH,1      ; AL$=INPUT$(1)
     INT 21H       ; PRINT AL$
;----------------------------------
     CMP AL,'Y'    ; IF AL$="Y" THEN
     JE YES        ; GOTO YES
     CMP AL,'y'    ; IF AL$="y" THEN
     JE YES        ; GOTO YES
;----------------------------------
     CMP AL,"N"    ; IF AL$="N" THEN
     JE NO         ; GOTO NO
     CMP AL,"n"    ; IF AL$="n" THEN
     JE NO         ; GOTO NO
;----------------------------------
     JMP START     ; GOTO START
;----------------------------------
YES:
     MOV AL,0      ; AL=0
     JMP BACK      ; GOTO BACK
;----------------------------------
NO:
     MOV AL,1       ; AL=1
BACK:
     MOV AH,4CH     ; Return AL(0,1)
     INT 21H        ; to Batch file
;----------------------------------
CODE ENDS                             ── CODEの部分のセグメントの終り
     END START                        ── プログラムの終り
```

（START: から INT 21H までの部分 ── プログラムそのもの）

このソース・プログラムの書き方は、COM ファイル専用のものです。それは、COM ファイルがメモリにロードされるときの構造が次のようになっているからです。

■COMファイル・ロード時のメモリ構成



- セグメントの値はすべて同じになる。
- セグメントの値は××××で示しているが，これはユーザーが設定するのではなく，MS-DOSがCOMファイルロード時に設定する。

- COMファイルは、プログラムが64Kバイト以内に納まるときに使います。すべてのセグメントが同じで開発しやすいという特長があります。そのため本書での例は、すべてCOMファイルにしています。
- COMファイルは、必ず100Hから開始しなければなりません。これは、アドレス0からの100Hバイトは、MS-DOSが作業領域に使うためです。

256バイト

0〜100H → 256B
0〜80H → 128B.
80〜100 → 128B

第2部

第3章 アセンブリ言語をさらに学ぶ

この章では、第１部第３章のプログラムYES NO. ASMを少し拡張して、もう少しアセンブリ言語を学ぶことにします。プログラミングの学習は、個々の命令を覚えるのではなく、あるまとまった処理をする命令の組み合わせをマスターしたほうが、実際の技術が身につくものです。

3・1 2者選択のプログラムを拡張する

YESNOプログラムを拡張するところは、次の2点です。

・Y／N以外のキーが押されたら、ビープ音を鳴らす。
・そのあと、「YかNで答えてください」というメッセージを表示する。

この2つは、MS-DOSに備わっている機能（ファンクション・コール）で実現することができます。わかりやすいように1つずつ拡張します。

■ビープ音を鳴らす

MS-DOSのファンクション・コールで、AHに2を入れ、DLにアスキーコードを入れて、INT 21Hを実行するとDLの文字を画面に表示することができます。

例えば、

●1文字表示のファンクション・コール。

```
MOV AH, 2   ；「A」を表示
MOV DL, 41 H；41Hは「A」のアスキーコード
INT 21 H
```

とすると、アスキーコード41Hは、「A」なので、Aが画面に表示されます。

そこで、DLにアスキーコード7を入れると、7はベル（BELL）なので、ビープ音が鳴ります。

```
MOV AH, 2   ；ビープ音を鳴らす
MOV DL, 7   ；7はベル
INT 21 H
```

これは、BASIC言語の、

```
PRINT CHR$(7)
```

と同じことです。
　これを追加したのが、次のプログラムです。

■「ビープ音」を追加したプログラム

```
;------------------------
;   YESNO2.ASM
; YES OR NO Input
; for Batch file
;------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
;--------------------------------
START:
     MOV AH,1      ; AL$=INPUT$(1)
     INT 21H       ; PRINT AL$
;--------------------------------
     CMP AL,'Y'    ; IF AL$="Y" THEN
     JE YES        ; GOTO YES
     CMP AL,'y'    ; IF AL$="y" THEN
     JE YES        ; GOTO YES
;--------------------------------
     CMP AL,"N"    ; IF AL$="N" THEN
     JE NO         ; GOTO NO
     CMP AL,"n"    ; IF AL$="n" THEN
     JE NO         ; GOTO NO
;--------------------------------
     MOV AH,2      ; PRINT CHR$(7)
     MOV DL,7
     INT 21H
;--------------------------------
     JMP START     ; GOTO START
;--------------------------------
YES:
     MOV AL,0      ; AL=0
     JMP BACK      ; GOTO BACK
;--------------------------------
NO:
     MOV AL,1       ; AL=1
BACK:
     MOV AH,4CH     ; Return AL(0,1)
     INT 21H        ; to Batch file
;--------------------------------
CODE ENDS
     END START
```

キー1文字入力と表示

「Y」,「y」ならYESへジャンプ

「N」,「n」ならNOへジャンプ

ビープ音を鳴らすプログラム

はじめに戻る

YESの処理

NOの処理

YES/NOに応じて0,1を返す

■「ビープ音」を追加したBASICプログラム

```
1    ' save "YESNO2.BAS"
100 '-----------------------
110 ' YESNO2 BASIC PROGRAM
120 '-----------------------
130 *START
140 AL$=INPUT$(1):PRINT AL$
150 '-----------------------
160 IF AL$="Y" THEN GOTO *YES
170 IF AL$="y" THEN GOTO *YES
180 IF AL$="N" THEN GOTO *NO
190 IF AL$="n" THEN GOTO *NO
195 PRINT CHR$(7)                    BEEP 1と同じ
200 GOTO *START
210 '-----------------------
220 *YES
230   AL=0
240 GOTO *BACK
250 '-----------------------
260 *NO
270 AL=1
280 '-----------------------
290 *BACK
300 PRINT AL
310 END
```

ソース・ファイル名をYESNO2.ASMとして、アセンブルし、リンクし、COMファイルに直すのが、次の操作です。

■COMファイル作成操作

● アセンブル

```
A>MASM YESNO2 ⏎
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

Object filename [YESNO2.OBJ]:⏎
Source listing  [NUL.LST]:⏎
Cross reference [NUL.CRF]:⏎

49694 Bytes free

Warning Severe
Errors  Errors
0       0
```

●リンク

```
A>LINK YESNO2 ⏎

Microsoft 8086 Object Linker
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985

Run File [YESNO2.EXE]:⏎
List File [NUL.MAP]:⏎
Libraries [.LIB]:⏎
Warning: no stack segment

A>
```

● COMファイル作成

```
A>EXE2BIN YESNO2 ⏎

A>
```

これでできた YESNO2. COM の実行例を次に示します。

```
A>YESNO2 ⏎
U
Y
A>YESNO2 ⏎        間違ったキーを入れるとビープ音が鳴る
M
Y
A>
```

これを組み込んだバッチ・ファイルは、次のとおりです。

■YESNO2を組み込んだバッチ・ファイル

```
ECHO OFF
CLS
ECHO ********** ファイル表示 **********
IF "%1"=="" GOTO END
:START
  ECHO ドライブBに表示したいディスクを入れてください.
  PAUSE
  ECHO ------------------------------------------------
  DIR B: ¦ FIND "%1"
REM --- YES/NO判断 ---
   ECHO 処理を続けますか(Y/N)?
   YESNO2
   IF ERRORLEVEL 1 GOTO END
   ECHO ------------------------------------------------
   GOTO START
:END
```

■メッセージを表示する

次に、ビープ音を鳴らしたあとに、警告のメッセージを表示してみましょう。

AH に 9 を入れて、DX にメッセージを格納している先頭アドレスを入れて、INT 21H を実行すると、メッセージを画面に表示することができます。

●文字列表示のファンクション・コール。

MOV AH, 9
MOV DX, メッセージ先頭アドレス
INT 21 H

ただし、メッセージはユーザーがメモリに格納し、その終わりにドルマーク($)を付けなくてはなりません。これは、MS-DOS の機能上、そういう決まりになっているからです。

これは、アセンブリ言語では次のようにします。

●文字列表示のファンクション・コールでは、文字列の先頭から文字を順次表示し、$にであうと表示を終了する。

MOV AH, 9
MOV DX, OFFSET MES
INT 21 H

MES DB 'Y か N で答えてください$'

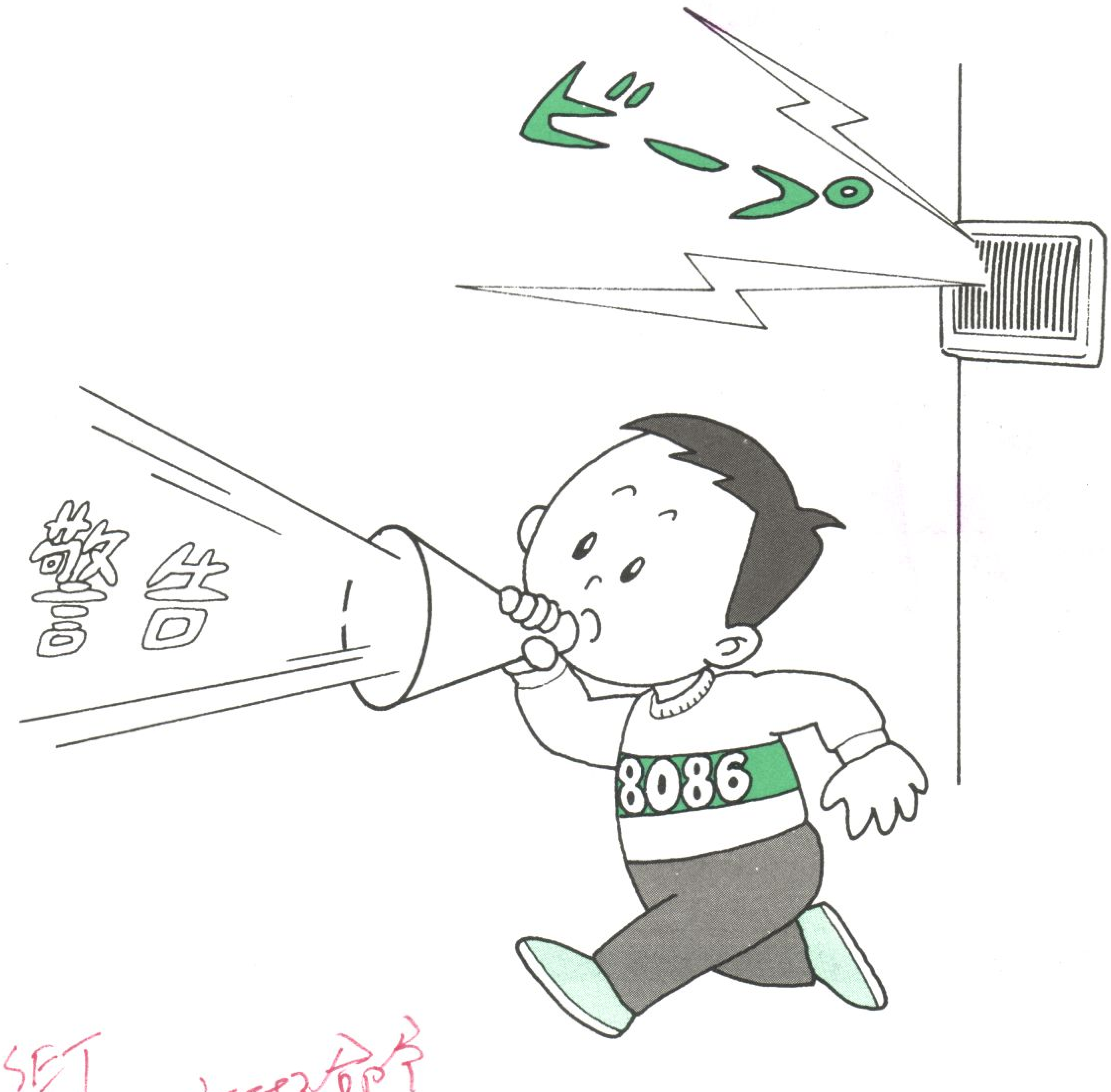
ラベルを MES (MESsage の略) として、メモリ上のアドレスを決

めます。実際のアドレスは、アセンブルのときに決定されます。

DB は、Define Byte のことで、バイト単位で DB 以下の文字列をメモリに格納する、アセンブラに対する命令です。8086 の命令ではありません。**クォート (') でくくると文字をそのままアスキーコードに直して、メモリに格納します。**

OFFSET MES は、ラベル MES のある先頭アドレスを表わすアセンブラに対する命令です。ここでは、こういうふうにするのだと覚えておいてください。

メッセージ表示を追加したのが、次のプログラムです。



■「メッセージ表示」を追加したプログラム

```
;-------------------------
;    YESNO3.ASM
; YES OR NO Input
; for Batch file
;-------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
;---------------------------------
START:
     MOV AH,1        ; AL$=INPUT$(1)
     INT 21H         ; PRINT AL$
;---------------------------------
     CMP AL,'Y'      ; IF AL$="Y" THEN
     JE YES          ; GOTO YES
     CMP AL,'y'      ; IF AL$="y" THEN
     JE YES          ; GOTO YES
;---------------------------------
     CMP AL,"N"      ; IF AL$="N" THEN
     JE NO           ; GOTO NO
     CMP AL,"n"      ; IF AL$="n" THEN
     JE NO           ; GOTO NO
;---------------------------------
     MOV AH,2        ; PRINT CHR$(7)
     MOV DL,7
     INT 21H
;---------------------------------
     MOV AH,9
     MOV DX,OFFSET MES
     INT 21H         ; PRINT
                     ; PRINT "YかNで答えてください"
                     ; PRINT
;---------------------------------
     JMP START       ; GOTO START
;---------------------------------
YES:
     MOV AL,0        ; AL=0
     JMP BACK        ; GOTO BACK
;---------------------------------
NO:
     MOV AL,1         ; AL=1
BACK:
     MOV AH,4CH       ; Return AL(0,1)
     INT 21H          ; to Batch file
;---------------------------------
MES  DB 13,10,'YかNで答えてください',13,10,'$'
CODE ENDS
     END START
```

メッセージ表示プログラム（MOV AH,9／MOV DX,OFFSET MES／INT 21H）

表示するメッセージをメモリに格納（MES DB ...）

メッセージをみやすくするため、キャリッジリターン、ラインフィードのコードを入れている（13,10）

■「メッセージ表示」を追加したBASICプログラム

```
1    ' save "YESNO3.BAS"
100 '------------------------
110 ' YESNO3 BASIC PROGRAM
120 '------------------------
130 *START
140 AL$=INPUT$(1):PRINT AL$
150 '------------------------
160 IF AL$="Y" THEN GOTO *YES
170 IF AL$="y" THEN GOTO *YES
180 IF AL$="N" THEN GOTO *NO
190 IF AL$="n" THEN GOTO *NO
195 PRINT CHR$(7)
197 PRINT "YかNで答えてください"
200 GOTO *START
210 '------------------------
220 *YES
230   AL=0
240 GOTO *BACK
250 '------------------------
260 *NO
270 AL=1
280 '------------------------
290 *BACK
300 PRINT AL
310 END
```

ここでソース・ファイル名をYESNO3. ASMとして、アセンブルし、リンクし、COMファイルに直さなければいけませんが、慣れてくると操作が面倒です。

そこで、ソース・ファイルをCOMファイルにする便利なバッチ・ファイルを紹介しましょう。

ファイル名： ASM. BAT

使　い　方： ASM ソース・ファイル名

内　　　容： MASM %1；
LINK %1；
EXE2BIN %1 %1. COM
DEL %1. OBJ
DEL %1. EXE

●COM ファイルができたあとは、OBJ ファイルと EXE ファイルは不要となるので削除してもよい。

これで、1 発で COM ファイルができ上がり、しかも不要のファイルが削除されます。%1 にソース・ファイル名が入ります。MASM と LINK のあとに、セミコロン（；）を付けると、各種生成ファイルのファイル名入力を省略することができます。

■アセンブルに便利なバッチ・ファイル

```
MASM %1;
LINK %1;
EXE2BIN %1 %1.COM
DEL %1.OBJ
DEL %1.EXE
```

次に実行例を挙げます。

■バッチ・ファイルでのアセンブル

```
A>ASM YESNO3 ⏎            ―― これ1行を入力するだけでよい

A>MASM YESNO3;
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984


49694 Bytes free

Warning Severe                          バッチ・ファイルの処理
Errors  Errors
0       0

A>LINK YESNO3;

Microsoft 8086 Object Linker
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985

Warning: no stack segment

A>EXE2BIN YESNO3 YESNO3.COM

A>DEL YESNO3.OBJ

A>DEL YESNO3.EXE

A>
```

これでできたYESNO3. COMの実行例を次に示します。

■YESNO3.COMの実行例

```
A>YESNO3 ⏎
U
 YかNで答えてください
Y                          間違ったキーを入れると
A>YESNO3                   メッセージが表示される
M
 YかNで答えてください
N
A>
```

これを組み込んだバッチ・ファイルは、次のとおりです。

■YESNO3 を組み込んだバッチ・ファイル

```
ECHO OFF
CLS
ECHO ********** ファイル表示 **********
IF "%1"=="" GOTO END
:START
  ECHO ドライブBに表示したいディスクを入れてください.
  PAUSE
  ECHO ---------------------------------------------
  DIR B: ¦ FIND "%1"
REM --- YES／NO判断 ---
   ECHO 処理を続けますか（Y／N）？
   YESNO3
   IF ERRORLEVEL 1 GOTO END
   ECHO ---------------------------------------------
   GOTO START
:END
```

ちなみにバッチ・ファイルの実行例を次に挙げます。

■バッチ・ファイル実行例

```
A>FDISP BAT

A>ECHO OFF
********** ファイル表示 **********
ドライブBに表示したいディスクを入れてください.
準備ができたらどれかキーを押してください.
---------------------------------------------
DIRSORT  BAT      474  87-07-10  17:45
MNU      BAT     1145  87-07-13  23:35
処理を続けますか（Y／N）？
M
YかNで答えてください   ―「N」を「M」と間違えて入れてメッセージが表示
N
A>
```

なお、マクロアセンブラの疑似命令でEQUを使って書いたソース・プログラムの例を次に挙げます。

■疑似命令を使った例

```
;--------------------------
;    YESNO4.ASM
;  YES OR NO Input
;  for Batch file
;--------------------------
INPUT    EQU  1
PRINT    EQU  2
BEEP     EQU  7
PSTRING  EQU  9
RETURN   EQU  4CH
Y        EQU  0
N        EQU  1
CR       EQU  13
LF       EQU  10
;--------------------------
CODE  SEGMENT
      ASSUME CS:CODE,DS:CODE
      ORG 100H
;----------------------------------
START:
      MOV AH,INPUT; AL$=INPUT$(1)
      INT 21H     ; PRINT AL$
;----------------------------------
      CMP AL,'Y'  ; IF AL$="Y" THEN
      JE YES      ; GOTO YES
      CMP AL,'y'  ; IF AL$="y" THEN
      JE YES      ; GOTO YES
;----------------------------------
      CMP AL,"N"  ; IF AL$="N" THEN
      JE NO       ; GOTO NO
      CMP AL,"n"  ; IF AL$="n" THEN
      JE NO       ; GOTO NO
;----------------------------------
      MOV AH,PRINT; PRINT CHR$(7)
      MOV DL,BEEP
      INT 21H
;----------------------------------
      MOV AH,PSTRING
      MOV DX,OFFSET MES
      INT 21H     ; PRINT
                  ; PRINT "YかNで答えてください"
                  ; PRINT
;----------------------------------
      JMP START   ; GOTO START
;----------------------------------
YES:
      MOV AL,Y    ; AL=0
      JMP BACK    ; GOTO BACK
;----------------------------------
NO:
      MOV AL,N     ; AL=1
BACK:
      MOV AH,RETURN ; Return AL(0,1)
      INT 21H       ; to Batch file
;----------------------------------
MES   DB CR,LF,'YかNで答えてください',CR,LF,'$'
CODE  ENDS
      END START
```

EQUで数値をラベルとして定義

数値の代りにラベルを使っている

冒頭で、EQU でラベルに定義した数字が、プログラムの中で意味のある表現に変わります。これでプログラムがわかりやすくなります。

これらのラベルが、アセンブルされて数値に直されたところをリストファイルでみると、次のようになっています。



■ラベルと実際の数値との関係

```
Microsoft MACRO Assembler  Version 3.00                  Page     1-1
                                                                  07-23-87

                                       ;------------------------
                                       ;   YESNO4.ASM
                                       ; YES OR NO Input
                                       ; for Batch file
                                       ;------------------------
= 0001                                 INPUT    EQU 1
= 0002                                 PRINT    EQU 2
= 0007                                 BEEP     EQU 7
= 0009                                 PSTRING  EQU 9
= 004C                                 RETURN   EQU 4CH
= 0000                                 Y        EQU 0
= 0001                                 N        EQU 1
= 000D                                 CR       EQU 13
= 000A                                 LF       EQU 10
                                       ;------------------------
0000                                   CODE SEGMENT
                                            ASSUME CS:CODE,DS:CODE
0100                                        ORG 100H
                                       ;--------------------------------
0100                                   START:
0100  B4 01                                 MOV AH,INPUT; AL$=INPUT$(1)
0102  CD 21                                 INT 21H     ; PRINT AL$
                                       ;--------------------------------
0104  3C 59                                 CMP AL,'Y'  ; IF AL$="Y" THEN
0106  74 1B                                 JE YES      ; GOTO YES
0108  3C 79                                 CMP AL,'y'  ; IF AL$="y" THEN
010A  74 17                                 JE YES      ; GOTO YES
                                       ;--------------------------------
010C  3C 4E                                 CMP AL,"N"  ; IF AL$="N" THEN
010E  74 18                                 JE NO       ; GOTO NO
0110  3C 6E                                 CMP AL,"n"  ; IF AL$="n" THEN
0112  74 14                                 JE NO       ; GOTO NO
                                       ;--------------------------------
0114  B4 02                                 MOV AH,PRINT; PRINT CHR$(7)
0116  B2 07                                 MOV DL,BEEP
0118  CD 21                                 INT 21H
                                       ;--------------------------------
011A  B4 09                                 MOV AH,PSTRING
011C  BA 012E R                        MOV DX,OFFSET MES
011F  CD 21                                 INT 21H     ; PRINT
                                                        ; PRINT "YかNで答えてください"
                                                        ; PRINT
                                       ;--------------------------------
0121  EB DD                                 JMP START   ; GOTO START
                                       ;--------------------------------
0123                                   YES:
0123  B0 00                                 MOV AL,Y    ; AL=0
0125  EB 03 90                              JMP BACK    ; GOTO BACK
                                       ;--------------------------------
0128                                   NO:
0128  B0 01                                 MOV AL,N     ; AL=1
012A                                   BACK:
012A  B4 4C                                 MOV AH,RETURN ; Return AL(0,1)
```

アセンブルされて数値に直っている

```
Microsoft MACRO Assembler  Version 3:00                  Page     1-2
                                                                  07-23-87

012C  CD 21                                 INT 21H        ; to Batch file
                                       ;--------------------------------
012E  0D 0A 82 78 82 A9MES  DB CR,LF,'YかNで答えてください',CR,LF,'$
                                       ;
      82 6D 82 C5 93 9A
      82 A6 82 C4 82 AD
      82 BE 82 B3 82 A2
      0D 0A 24
0149                                   CODE ENDS
                                            END START
```

3・2 メニュー選択をするプログラムを作る

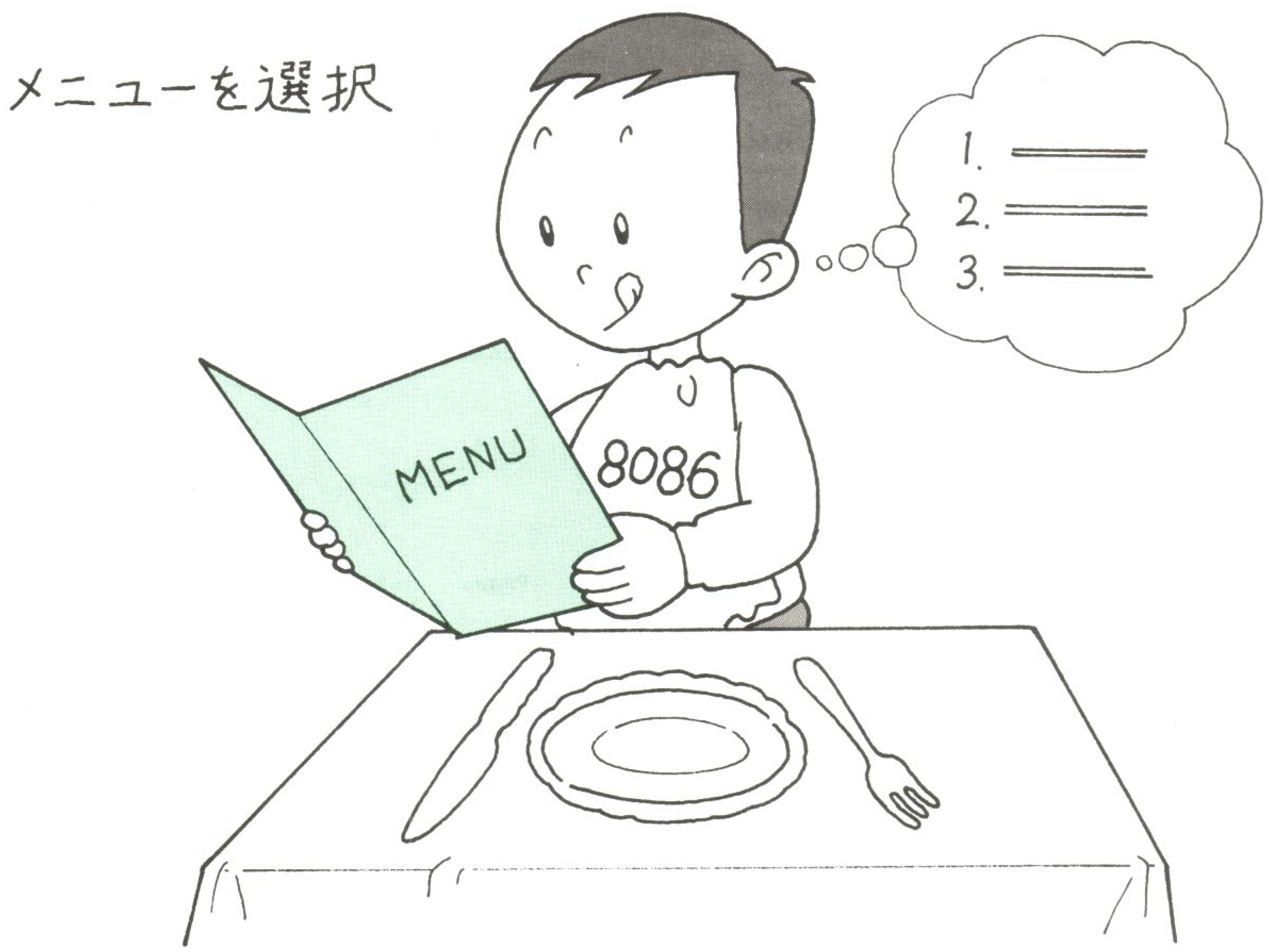
YES／NOの2者選択プログラムを発展させ、IF ERRORLEVELの条件判断を使って、いくつかの処理項目（メニュー）から選択するようにすることもできます。

例を2つ、挙げます。まずは、3つの項目から選択するものです。

メニュー例：

■3つの項目のメニュー

```
---  ディレクトリ・ソート  ---
1...ファイル名順
2...日付け順
3...MS-DOS
-----------------------------
処理の番号を選んでください。
```



メニュー選択例：

■3個のメニュー選択例

```
A>DIRSORT ⏎

A>ECHO OFF
---  ディレクトリ・ソート ---
1...ファイル名順
2...日付け順
3...MS-DOS
------------------------------
処理の番号を選んでください。
2-----------------------------
FILES     BAT       657  87-07-09   1:59
FTYPE     BAT       427  87-07-13  23:26
FDISP     BAT       413  87-07-14   1:10
MACHINE   BAK      2833  87-07-14  14:38
MACHINE   2        8083  87-07-17  17:13
SYSTEM    JXW      6073  87-07-18  16:59
MACHINE            7039  87-07-22  16:11
MACHINE   1        7919  87-07-22  16:31
PART_I        <DIR>      87-07-22  18:48
PART_II       <DIR>      87-07-24   4:54
準備ができたらどれかキーを押してください.
---  ディレクトリ・ソート ---
1...ファイル名順
2...日付け順
3...MS-DOS
------------------------------
処理の番号を選んでください。
3-----------------------------

A>
```

バッチ・ファイルの実行

メニュー表示

2を選択

日付け順にソートされている

メニュー表示

3を選択

バッチ・ファイルの内容：

■3つの項目選択のバッチ・ファイル

```
:START
ECHO OFF
CLS
    ECHO ---  ディレクトリ・ソート ---
    ECHO 1...ファイル名順
    ECHO 2...日付け順
    ECHO 3...ＭＳ－ＤＯＳ
    ECHO ------------------------------
    ECHO 処理の番号を選んでください。
    INPUT ——— キー入力により番号を返すマシン語プログラム
    ECHO ------------------------------
REM --- 分岐処理 ---
    IF ERRORLEVEL 3 GOTO MSDOS       ┐ 判断と分岐
    IF ERRORLEVEL 2 GOTO DATE        ┘
REM ---- ファイル名順 ---
    DIR B: ¦ FIND "-" ¦ SORT         ┐
    PAUSE                            │ ①の処理
    GOTO START                       │
:DATE                                ┘
REM --- 日付け順 ---
    DIR B: ¦ FIND "-" ¦ SORT/+25     ┐
    PAUSE                            │ ②の処理
    GOTO START                       │
:MSDOS                               ┘
REM --- ＭＳ－ＤＯＳ ---             ——— ③の処理
```

次は、6つの項目から選択するものです。

メニュー例：

■6つの項目のメニュー

```
-------------- メニュー選択 --------------------------------
1...データディスク作成 (FORMAT B:)
2...システムディスク作成 (FORMAT B:/S)
3...高速データ消去 (FORMAT B:/C)
4...ディスク初期化＋まるごと複写 (FORMAT B:+DISKCOPY A: B:)
5...すべてのファイル複写 (COPY A:*.* B:)
6...ＭＳ－ＤＯＳ
------------------------------------------------------------
  処理の番号を選んでください。
```

メニュー選択例：

■6個のメニュー選択例

```
A>MNU ⏎ ——バッチ・ファイル

A>ECHO OFF
-------------- メニュー選択 --------------------------------
1...データディスク作成 (FORMAT B:)
2...システムディスク作成 (FORMAT B:/S)
3...高速データ消去 (FORMAT B:/C)                          メニュー表示
4...ディスク初期化+まるごと複写 (FORMAT B:+DISKCOPY A: B:)
5...すべてのファイル複写 (COPY A:*.* B:)
6...MS-DOS
------------------------------------------------------------
  処理の番号を選んでください。
5-------------------------- —— 5を選択
A:ASM.BAT
A:FDISP.BAT
A:DIRSORT.BAT
A:FTYPE.BAT         — ファイルがコピーされる
A:MNU.BAT
A:INPUT2.COM
        6 個のファイルをコピーしました.
-------------- メニュー選択 --------------------------------
1...データディスク作成 (FORMAT B:)
2...システムディスク作成 (FORMAT B:/S)
3...高速データ消去 (FORMAT B:/C)                          メニュー表示
4...ディスク初期化+まるごと複写 (FORMAT B:+DISKCOPY A: B:)
5...すべてのファイル複写 (COPY A:*.* B:)
6...MS-DOS
------------------------------------------------------------
  処理の番号を選んでください。
6-------------------------- —— 6を選択

A>
```

バッチ・ファイルの内容：

■6つの項目選択のバッチ・ファイル

```
ECHO OFF
:START
CLS
   ECHO -------------- メニュー選択 ------------------------------
   ECHO 1...データディスク作成 (FORMAT B:)
   ECHO 2...システムディスク作成 (FORMAT B:/S)
   ECHO 3...高速データ消去 (FORMAT B:/C)
   ECHO 4...ディスク初期化+まるごと複写 (FORMAT B:+DISKCOPY A: B:)
   ECHO 5...すべてのファイル複写 (COPY A:*.* B:)
   ECHO 6...MS-DOS
   ECHO ------------------------------------------------------------
   ECHO  処理の番号を選んでください。 キー入力により番号を返すマシン語プログラム
   INPUT2
   ECHO ---------------------------
REM --- 分岐処理 ---
   IF ERRORLEVEL 6 GOTO MSDOS        ┐
   IF ERRORLEVEL 5 GOTO COPY         │
   IF ERRORLEVEL 4 GOTO DISKCOPY     │ 判断と分岐
   IF ERRORLEVEL 3 GOTO FORMATBC     │
   IF ERRORLEVEL 2 GOTO FORMATBS     ┘
REM --- データディスク作成 ---
   FORMAT B:                          ┐ ①の処理
   GOTO START                         ┘
REM ---- システムディスク作成 ---
:FORMATBS                             ┐
   FORMAT B:/S                        │ ②の処理
   GOTO START                         ┘
REM --- 高速データ消去 ---
:FORMATBC                             ┐
   FORMAT B:/C                        │ ③の処理
   GOTO START                         ┘
REM -- ディスク初期化+まるごと複写 ---
:DISKCOPY                             ┐
   FORMAT B:                          │ ④の処理
   DISKCOPY A: B:                     │
   GOTO START                         ┘
REM --- すべてのファイル複写 ---
:COPY                                 ┐
   COPY A:*.* B:                      │ ⑤の処理
   GOTO START                         ┘
REM --- MS-DOS ---
:MSDOS                                ── ⑥の処理
```

● IF　ERRORLEVEL では、エラーコードの数の多いほうから判定する。

なお、IF ERRORLEVEL での判断では、数字の多いほうから判定をする必要があります。これは、その数字以上の値で「真」になる（条件を満たす）ようになっているからです。

例えば、

IF ERRORLEVEL 3

では、3以上の値がすべて「真」になります。

入力した数字が3のときに、処理Bに行きたい場合、

IF ERRORLEVEL 2 GOTO A
IF ERRORLEVEL 3 GOTO B

とすると、はじめの判定で2以上の数字なので、「真」になり、処理A に行き、処理Bには行かないことになります。

■ 3つの項目のメニュー・プログラム

3つの項目のメニューのバッチ・ファイルで、INPUTというのが、マシン語ファイル(INPUT.COM)で、プログラムの処理内容は、次のようにしています。

- キーボードから入力を受け付け、その内容を表示する。
- 入力されたものが1, 2, 3以外であれば、再び入力させる。
- 入力された文字1, 2, 3に対応して、コード(数値)を1, 2, 3と出し、バッチ・ファイルに与える。

これをBASIC言語で表現すると、次のようになります。

■1, 2, 3を表示するBASICプログラム

```
1    ' save "INPUT.BAS"
100 '------------------------
110 ' INPUT BASIC PROGRAM
120 '------------------------
130 *START
140 AL$=INPUT$(1):PRINT AL$
150 '------------------------
160 IF AL$<="0" THEN GOTO *START
170 IF AL$>="4" THEN GOTO *START
180 '------------------------
190 *PROCESS
200   AL=ASC(AL$)-&H30
210   PRINT AL
220 '------------------------
230 END
```

(160–170) 文字の入力をチェック 1−3以外は入力のしなおし

(200) 文字1, 2, 3を数値1, 2, 3に直す

なお、ここでも、次のファイルが必要です。

- MASM. EXE
- LINK. EXE
- EXE2BIN. COM

以下、プログラムの作成・開発手順を説明します。

1. INPUT. ASM というファイル名として、次のように書きアセンブリ言語プログラムを作ります。

■1, 2, 3を返すアセンブリ言語プログラム

```
;-------------------------
;    INPUT.ASM
; Parameters Input
; for Batch file
;-------------------------
INPUT   EQU 1
RETURN EQU 4CH
;-------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
START:
     MOV AH,INPUT; AL$=INPUT$(1)
     INT 21H      ; PRINT AL$
;----------------------------------
     CMP AL,'0'   ; IF AL<=0 THEN
     JBE START    ; GOTO START
     CMP AL,'4'   ; IF AL>=4 THEN
     JAE START    ; GOTO START
;----------------------------------
PROCESS:
     SUB AL,30H      ; AL=AL-30H
     MOV AH,RETURN   ; Return AL(1-3)
     INT 21H         ; to Batch file
;----------------------------------
CODE ENDS
     END START
```

文字の入力チェック
1−3以外は入力のしなおし

文字1, 2, 3を数値
1, 2, 3に直して返す

2. 次のようにして INPUT. ASM をアセンブルし、COM ファイルにします。

A＞ASM INPUT

■COMファイル作成操作

```
A>ASM INPUT ⏎

A>MASM INPUT;
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984


49698 Bytes free

Warning Severe
Errors  Errors
0       0

A>LINK INPUT;

Microsoft 8086 Object Linker
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985

Warning: no stack segment

A>EXE2BIN INPUT INPUT.COM

A>DEL INPUT.OBJ

A>DEL INPUT.EXE

A>
A>
```

なお、アセンブル用バッチ・ファイルの内容は、次のとおりです。

■アセンブル用バッチ・ファイル

```
MASM %1;
LINK %1;
EXE2BIN %1 %1.COM
DEL %1.OBJ
DEL %1.EXE
```

これで、コードをバッチ・ファイルに渡すINPUT. COMというコマンド・ファイルができました。バッチ・ファイルを実行してみてください。

■アセンブリ言語プログラムの説明

基本的な内容は、YES／NOの選択のものと変わりがありません。ここで新しくでてきた命令について説明します。

●1文字入力・表示のファンクション・コール。

```
MOV AH, 1
INT 21 H
```

で、1文字をキーボードから入力したあと、次のような処理をしています（ALには、入力された文字のアスキーコードが入っています）。

●文字の比較とその結果に応じたジャンプ。

```
CMP AL, '0'
JBE START
```

は、ALの内容を文字の0（文字0のアスキーコード）と比較して、「0より大きいかまたは等しいならば、ラベルSTARTにジャンプする」ということです。

JBEは、Jump on Below or Equalの略で、below(下)またはequal（等しい）ときにジャンプする命令です。

これで、入力されたキーが0以下であれば、もう1度入力のしなおしをすることになります。BASIC言語では、次のようになります。

```
IF AL$<="0"  THEN GOTO *START
```

```
CMP AL, '4'
JAE START
```

は、ALの内容を文字の4（文字0のアスキーコード）と比較して、「4より大きいかまたは等しいならば、ラベルSTARTにジャンプする」ということです。

JAEは、Jump on Above or Equalの略で、above(上)またはequal（等しい）ときにジャンプする命令です。

これで、入力されたキーが4以上であれば、もう1度入力のしなおしをすることになります。BASIC言語では、次のようになります。

```
IF AL$>= "4"  THEN GOTO *START
```

●SUBは、引き算をする命令。

```
SUB AL, 30H
```

は、ALの内容から16進の30を引いて、ALに入れることです。

SUBは、SUBtract（引く）という意味です。BASIC言語では、次のようになります。

AL＝AL－&H30

なぜ、ここでALの内容から30Hを引いているかといいますと、入力された数字は文字扱いになっているので、数値に直しているからです。ほら、アスキーコード表で、文字の1は31H,2は32H,3は33Hとなっており、それぞれ30Hを引けば数値の1,2,3となるでしょう。

そして結果（ALの内容）を、

```
MOV AH, 4CH
INT 21 H
```

●結果をバッチ・ファイルに戻すファンクション・コール。

で、バッチ・ファイルに渡しています。

■ 6つの項目のメニュー・プログラム

次に、プログラムを拡張して、6つの項目のメニュー・プログラムを作ります。拡張するのは、次の3点です。

- 入力できる数を6までにする。
- 1−6以外のキーが押されたら、ビープ音を鳴らす。
- そのあと、「1から6までの数を入れてください」というメッセージを表示する。

ビープ音とメッセージ表示は、前出と同じくMS-DOSに備わっている機能を利用します。

<ビープ音を鳴らす>

●1文字表示のファンクション・コール。

```
MOV AH, 2
MOV AL, 7
INT 21 H
```

<メッセージを表示する>

●文字列表示のファンクション・コール。

```
MOV AH, 9
MOV DX, OFFSET MES
INT 21 H
MES DB '1から6までの数を入れてください$'
```

以上の点を追加したのが、次のプログラムです。

■1－6の数字を返すアセンブリ言語プログラム

```
;-------------------------
;   INPUT2.ASM
; Parameters Input
; for Batch file
;-------------------------
INPUT   EQU 1
PRINT   EQU 2
SOUND   EQU 7      ── BEEPは飛び先のラベルで使っているので
                      変更している(同じラベル名は使えない)
PSTRING EQU 9
RETURN  EQU 4CH
MIN     EQU '0'
MAX     EQU '7'
CR      EQU 13
LF      EQU 10
;-------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
START:
     MOV AH,INPUT; AL$=INPUT$(1)
     INT 21H     ; PRINT AL$
;-----------------------------------
     CMP AL,MIN  ; IF AL<=0 THEN
     JBE BEEP    ; GOTO BEEP ──── 1－6以外のキーなら
     CMP AL,MAX  ; IF AL>=7 THEN      BEEPに飛ぶ
     JAE BEEP    ; GOTO BEEP
;-----------------------------------
PROCESS:
     SUB AL,30H      ; AL=AL-30H
     MOV AH,RETURN   ; Return AL(1-6)
     INT 21H         ; to Batch file
;-----------------------------------   ── ビープ音
BEEP:
     MOV AH,PRINT
     MOV DL,SOUND    ; PRINT CHR$(7)     ── メッセージ表示
     INT 21H
;-----------------------------------
MESSAGE:
     MOV AH,PSTRING
     MOV DX,OFFSET MES
     INT 21H      ; PRINT
                  ; PRINT "1から6までの数を入れてください"
                  ; PRINT
     JMP START    ; GOTO START
;-----------------------------------
MES DB CR,LF,'1から6までの数を入れてください',CR,LF,'$'
CODE ENDS
     END START                      ── メッセージ
```

これを BASIC 言語で表現したのが、次のリストです。

■1－6の数字を表示するBACICプログラム

```
1     ' save "INPUT2.BAS"
100 '-----------------------
110 ' INPUT2 BASIC PROGRAM
120 '-----------------------
130 *START
140 AL$=INPUT$(1):PRINT AL$
150 '-----------------------
160 IF AL$<="0" THEN GOTO *ALARM
170 IF AL$>="7" THEN GOTO *ALARM
180 '-----------------------
190 *PROCESS
200   AL=ASC(AL$)-&H30
210   PRINT AL
220 '-----------------------
230 END
240 *ALARM
250 PRINT CHR$(7)
260 PRINT "1から6までの数を入れてさい"
270 GOTO *START
```

1－6以外のキーならALARMに飛ぶ（BEEPはBASICの命令なのでラベルには使えない）

ビープ音

メッセージ表示

このCOMファイルをバッチ・ファイルを実行して次のように作ってみてください。

■COMファイル作成用バッチファイル

```
MASM %1;
LINK %1;
EXE2BIN %1 %1.COM
DEL %1.OBJ
DEL %1.EXE
```

■アセンブル

```
A>
A>ASM INPUT2 ⏎

A>MASM INPUT2;
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984


49640 Bytes free

Warning Severe
Errors  Errors
0       0

A>LINK INPUT2;

Microsoft 8086 Object Linker
Version 3.00 (C) Copyright Microsoft Corp 1983, 1984, 1985

Warning: no stack segment

A>EXE2BIN INPUT2 INPUT2.COM

A>DEL INPUT2.OBJ

A>DEL INPUT2.EXE

A>
```

第3部 マシン語をアセンブラで開発する

マシン語でプログラミングができると、アプリケーション・ソフトと性質の違うプログラム開発ができます。それは、ユーティリティ・プログラムやシステムの機能拡張をするプログラムです。 第3部では、これまで得たマシン語やアセンブラの知識をベースにして、実用的なプログラム開発を行い、アセンブラでのプログラミング技法をさらにマスターします。

第3部

第1章 ターゲットマシンとOSについての知識

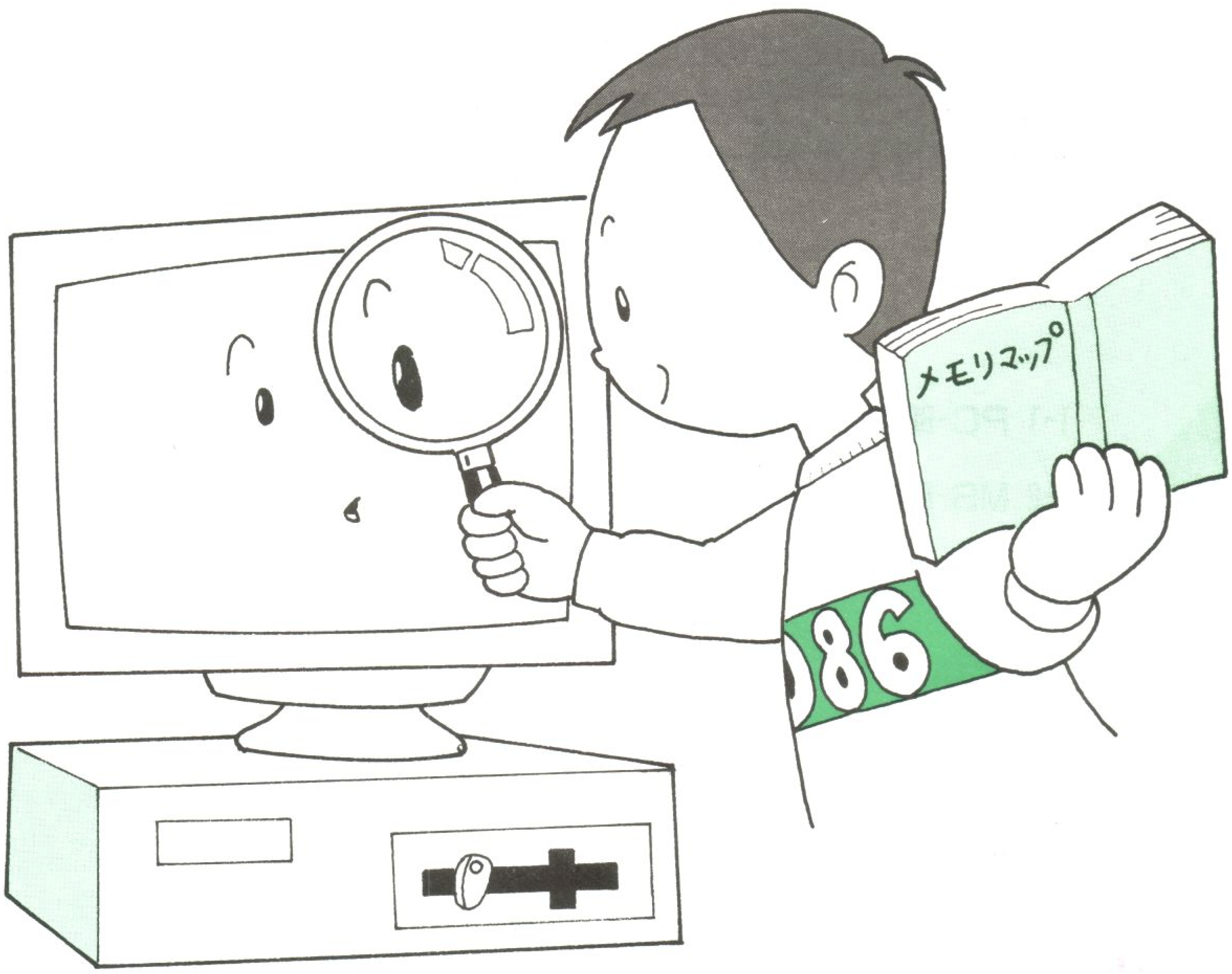
この章では、プログラムを開発するマシン(PC-98)とそのOS(MS-DOS)について、プログラム開発に必要な情報のほんのさわりを紹介します。

1·1 PC-9800シリーズのメモリ・マップ

マシン語プログラムでもBASICプログラムでも、プログラムを開発するのに使用するマシンと開発したプログラムを実行するマシンとがあります。

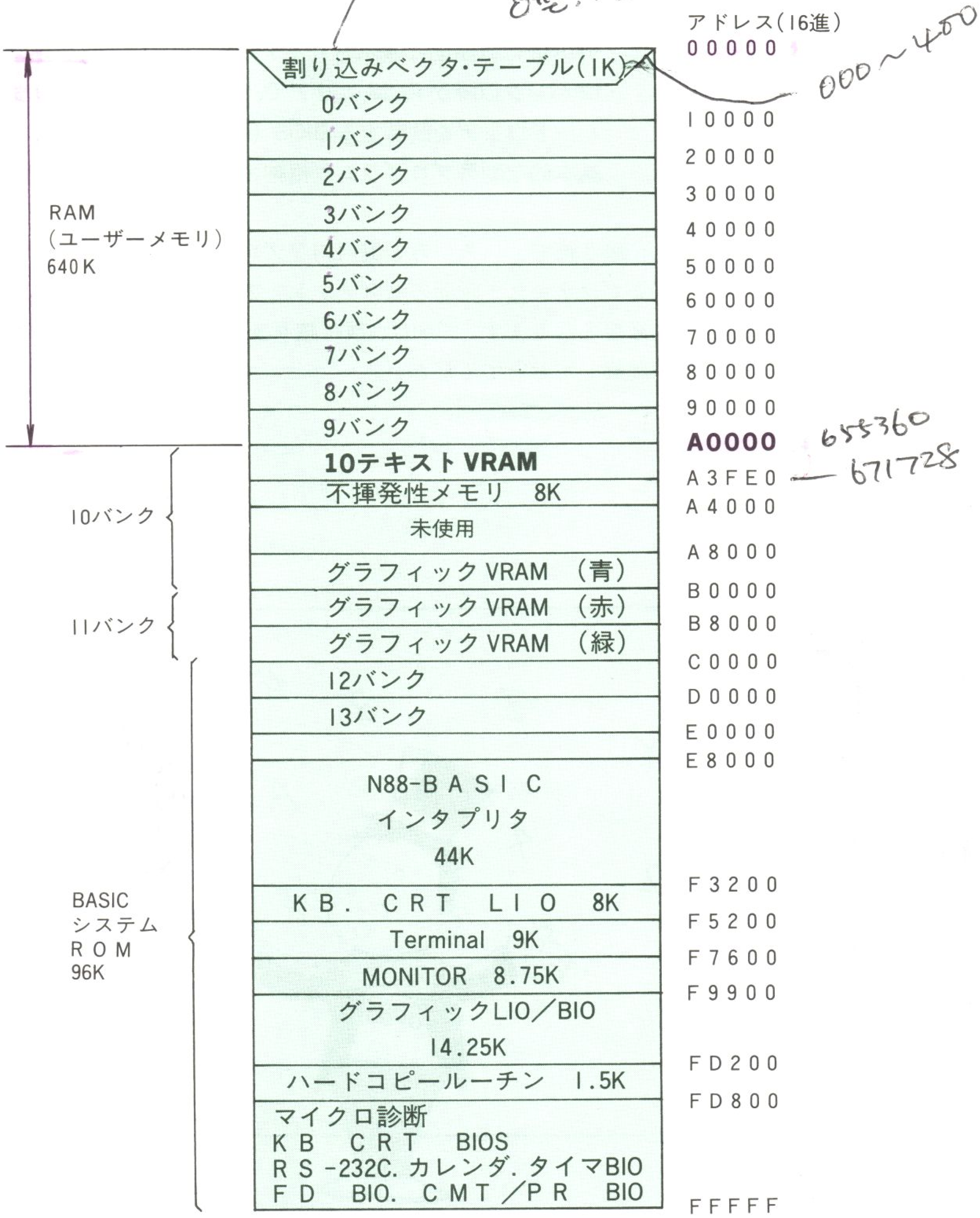
プログラムを開発するマシンを**開発マシン**というなら、プログラムを実行するマシンは目的のマシンということで**ターゲットマシン**といいます。普通、パソコンのユーザーでは、開発マシンとターゲットマシンは、同じです。

マシン語プログラム開発にあたっては、ターゲットマシンの内部構造（特にメモリ・マップ）とそのOSの構造や機能をよく知っておく必要があります。ここで、PC-9800シリーズを例にして、そのメモリ・マップ（システムがどのようにメモリに割り振られているかを示すもの）を紹介します。



■98のメモリ・マップ

■98のメモリ・マップ



※説明の便宜上、アドレスと図の比率は正確ではありません。

・図の1バンクとは、64Kバイトの領域を指します。

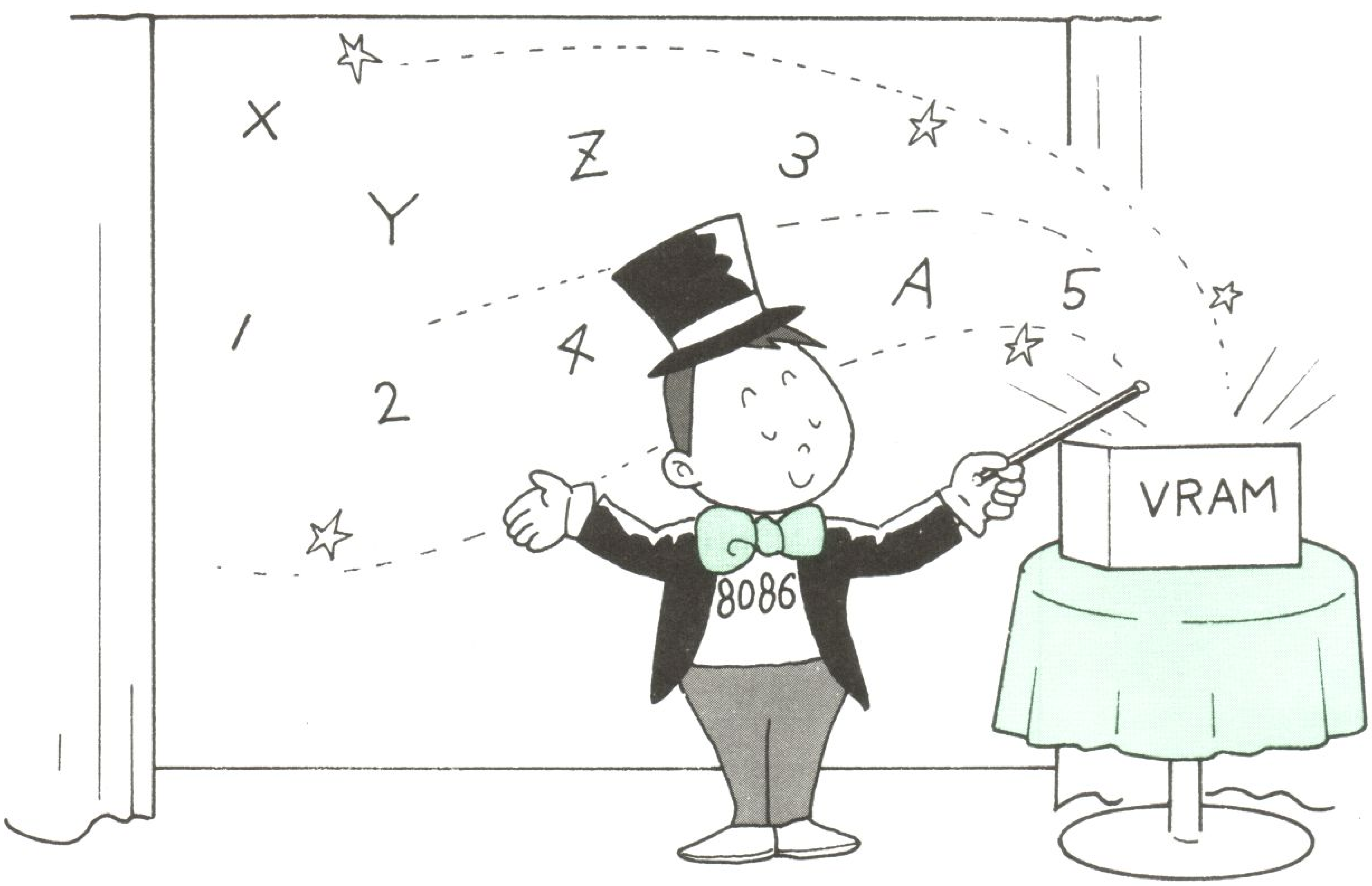
●ベクタ・テーブルには、INT命令で呼び出される**プログラムの開始アドレス**が入っている。

●PC—98では、文字を表示するテキスト画面とグラフィックスを表示するグラフィック画面は別になっている。

●BIOSは、バイオスと読み話題の互換機作りのかなめとなる部分。

- 0バンクのはじめの1Kバイトにソフトウェア割り込みのアドレス（ベクタ・テーブル）があります。
- 9バンクまでがユーザーエリアで640Kバイトあります。
- 10と11バンクがテキスト画面とグラフィック画面を表示するRAM（Video RAMといいVRAMと略す）があります。
- VRAMには画面に表示されるデータが入ります。
- 12バンク以降がROMエリアで、BASICのインタプリタをはじめ、ハードウェアを制御するBIOS（基本入出力システム）など98の基本的な制御プログラムが格納されています。

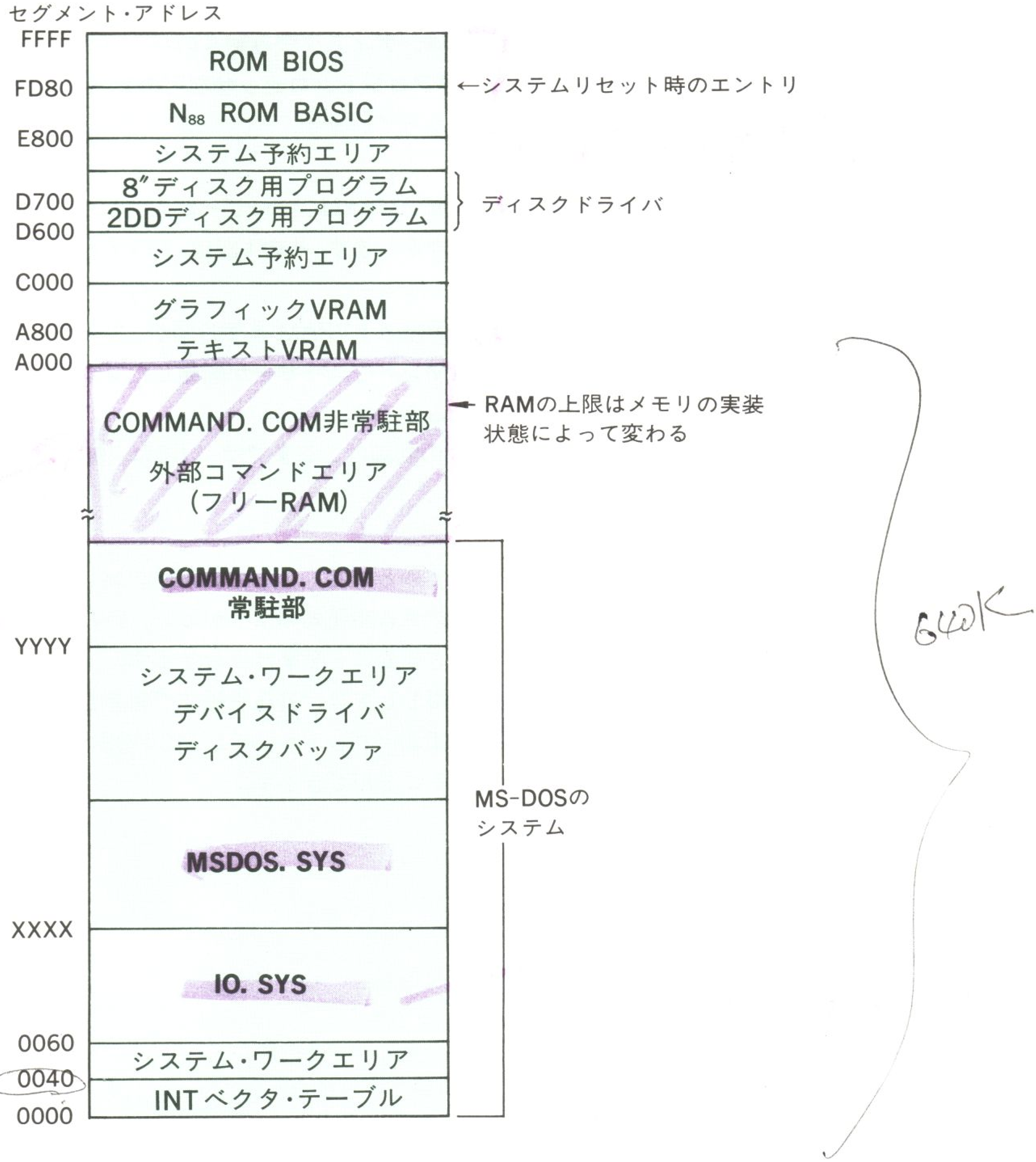
第3部で、いろいろと実用的なプログラムを紹介していますが、そこではテキスト画面のセグメント・アドレスやその構造などの知識が必要となります。詳細は、後に譲りますが、ここでは98全体の概要を理解しておいてください。



1・2 MS-DOSのメモリ・マップ

また、PC-9800シリーズでMS-DOSを起動したときのメモリ・マップは、次のようになっています。

■MS-DOSのシステム・メモリ・マップ



- 今度は、アドレスの小さいほうを下にして書いています（こういう書き方もあります）。
- アドレスXXXX, YYYYは起動した環境によって異なります。

・INT ベクタ・テーブルは 8086 の基本的なものですので同じです (その内容は違いますが)。
・その次に、MS-DOS が使うワークエリアがあります。
・IO. SYS は、Input ／ Output SYStem で、98 のハードウェアの制御を行うものです。
・MSDOS. SYS は、MS-DOS のシステムの中核となる部分で MS-DOS の機能を統括・処理するものです。
・それ以降は、ディスク読み書きのバッファや、機能を追加したもの(日本語フロントプロセッサなどのデバイスドライバ)、ワークエリアとなっています。

● IO.SYS(アイオーシス)、MSDOS.SYS(エムエスドスシス),COMMAND.COM(コマンドコム)は、MS-DOS の 3 本柱。

・COMMAND. COM は、ユーザーが入力したコマンドを解釈し、実行する働きをします。その 1 部は、YYYY から常にメモリにあります。
・以降からテキスト VRAM までの間が、ユーザーエリアで MS-DOS の外部コマンドやユーザーが開発したプログラムをロードし、実行するところです。
・MS-DOS の制御はここまでで、以降は、98 の ROM のシステムです。

この章のまとめとして、ターゲットマシンと OS についての知識があれば、プログラム開発効率は上がるし、いろいろなプログラムを開発することができることを挙げておきます。

この種の情報としては、書籍や雑誌で紹介されているシステムの解析情報に頼ればよいでしょう。98 と MS-DOS であれば、すぐに豊富な資料を入手することができます。

第3部

第2章 実用プログラムの開発

この章では、どんな操作にも良く使うファンクションキーの内容をコマンド実行だけで設定したり、より使いやすいファンクションキーにしたりする、実用的なプログラムを開発します。その過程で、8086の基本的で重要な命令をマスターできるように解説します。

2・1 ファンクションキー内容設定コマンドを作る

MS-DOSを起動したときには、ファンクションキーが10個表示されています。この10個は、コマンド入力の際の修正を行うための機能となっています。

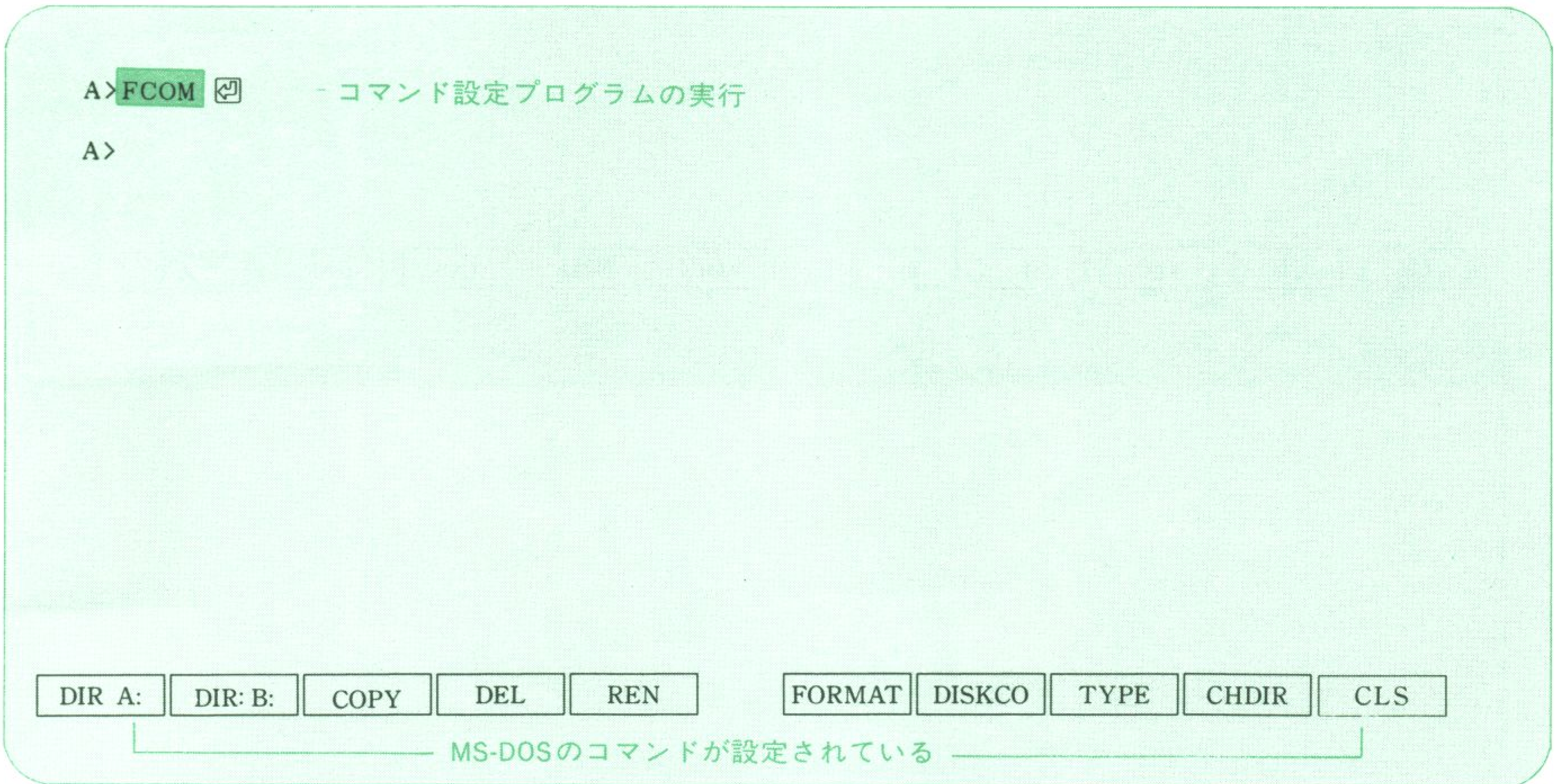
この10個に、自分の好きな内容を設定するコマンドを作りましょう。すると、次のメリットが生じます。

- KEYコマンドでファンクションキーの内容を設定するより、操作が簡単。コマンドを入力するだけでOK。
- KEYコマンドと違い、10個一度に設定できる。
- DIRやTYPEなどMS-DOSのコマンドを入力したいとき、SHIFTを押しながら、ファンクションキーを押す必要がない。
- 自分の好きな内容がいろいろとコマンドで設定できる。

2·2 設定の実行例

よく使う MS-DOS のコマンドを設定したのが、次の例です。FCOM.COM というファイルで、Function Keys に MS-DOS の COMmand を設定するという意味で付けました。

■MS-DOSコマンドの設定



もちろん、ファンクションキーの内容をもとのものに設定しなおすコマンドも作ります。FINIT. COM というファイルで Function Keys の INITialize（初期化）という意味で付けました。

■初期設定に戻す

```
A>FINIT ⏎ ——— 初期設定コマンドの実行

A>




 C1   CU   CA   S1   SU     VOID  NML  INS  REP  ^Z
    └──────────── もとに戻っている ────────────┘
```

2·3 ファンクションキーの設定法

ファンクションキーの内容を設定するには、次の2つのことを知っておく必要があります。

1. **ファンクションキーの内容を設定するファンクション・コール。**
2. **ファンクションキーのメモリへの格納状態。**

以下、この2点について説明します。

■ 設定のファンクション・コール

ファンクションキーの内容を設定するファンクション・コールは、次のとおりです。

DX ←設定する内容をメモリに格納して、その先頭アドレスを入れる。

CL ← 0DH（設定コマンド）

AX ←1, 2, . . . , 9, 10（キーの番号）

INT 0DCH ；ファンクションキー読み書きの
ファンクション・コール

例えば、f･1に「TYPE　」を設定するには、次のようにします。

Int 0DCH
AH ← 0DH

■ f･1 に「TYPE　」を設定

```
;--------------------------------------
START:
        MOV DX,OFFSET BUFF  ; Fkeys Addr
        MOV CL,0DH          ; Fkeys Write
        MOV AX,1            ;
        INT 0DCH            ; f·1 Write
;--------------------------------------
        MOV AH,0
        INT 21H           ; Return to DOS
;--------------------------------------
BUFF    DB 'TYPE  '         ; f·1
        DB 11 DUP (0)
```

「TYPE　」を格納しているアドレス

メモリに「TYPE　」と00を11個格納

BUFF（バッファの意）のところで、設定する内容「TYPE　」を格納していますが、その先に、まだ何か命令が続いています。なぜ、こういうふうにしているかは、次で述べるファンクションキーの格納状態を調べて、それと同じ形になるようにする必要があるからです。

なお、このINT 0DCHは、MS-DOSが内部で使用しているもので、INT 21Hのようにユーザーに公開されているものではありません。それでも、ユーザープログラムから利用することができます。

■ファンクションキーの格納状態

本書の冒頭で、ファンクションキーの内容を読み取りましたが、ここでもう1度、読み取ってみましょう。

ファンクションキーの内容を読み取るファンクション・コールは、次のとおりです。

DX ←読み取る内容を格納するメモリの先頭アドレスを入れる。

CL ← 0CH（読み取りコマンド）

AX ← 0（すべてのキーの内容を読み取る）

INT 0DCH ； ファンクションキー読み書きの
ファンクション・コール

次のように、入力してください。

■ファンクションキーの内容読み取り

●読み取りプログラムの入力と実行

```
A>SYMDEB ⏎
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-A ⏎
1C30:0100 MOV DX,200 ⏎
1C30:0103 MOV CL,C ⏎
1C30:0105 MOV AX,0 ⏎
1C30:0108 INT DC ⏎
1C30:010A ⏎
-G10A ⏎
AX=0000  BX=0000  CX=000C  DX=0200  SP=E2BF  BP=0000  SI=0000  DI=0000
DS=1C30  ES=1C30  SS=1C30  CS=1C30  IP=010A   NV UP EI PL NZ NA PO NC
1C30:010A 0000          ADD     [BX+SI],AL                    DS:0000=CD
-
```

● 読み取った結果

```
-D200,33F ⏎
1C30:0200  FE 20 43 31 20 20 1B 53-00 00 00 00 00 00 00 00   ~ C1  .S........
1C30:0210  FE 20 43 55 20 20 1B 54-00 00 00 00 00 00 00 00   ~ CU  .T........
1C30:0220  FE 20 43 41 20 20 1B 55-00 00 00 00 00 00 00 00   ~ CA  .U........
1C30:0230  FE 20 53 31 20 20 1B 56-00 00 00 00 00 00 00 00   ~ S1  .V........
1C30:0240  FE 20 53 55 20 20 1B 57-00 00 00 00 00 00 00 00   ~ SU  .W........
1C30:0250  FE 56 4F 49 44 20 1B 45-00 00 00 00 00 00 00 00   ~VOID .E........
1C30:0260  FE 4E 57 4C 20 20 1B 4A-00 00 00 00 00 00 00 00   ~NWL  .J........
1C30:0270  FE 49 4E 53 20 20 1B 50-00 00 00 00 00 00 00 00   ~INS  .P........
1C30:0280  FE 52 45 50 20 20 1B 51-00 00 00 00 00 00 00 00   ~REP  .Q........
1C30:0290  FE 20 5E 5A 20 20 1B 5A-00 00 00 00 00 00 00 00   ~ ^Z  .Z........
1C30:02A0  64 69 72 20 61 3A 0D 00-00 00 00 00 00 00 00 00   dir a:..........
1C30:02B0  64 69 72 20 62 3A 0D 00-00 00 00 00 00 00 00 00   dir b:..........
1C30:02C0  63 6F 70 79 20 00 00 00-00 00 00 00 00 00 00 00   copy ...........
1C30:02D0  64 65 6C 20 00 00 00 00-00 00 00 00 00 00 00 00   del ............
1C30:02E0  72 65 6E 20 00 00 00 00-00 00 00 00 00 00 00 00   ren ............
1C30:02F0  63 68 6B 64 73 6B 20 61-3A 0D 00 00 00 00 00 00   chkdsk a:.......
1C30:0300  63 68 6B 64 73 6B 20 62-3A 0D 00 00 00 00 00 00   chkdsk b:.......
1C30:0310  74 79 70 65 20 00 00 00-00 00 00 00 00 00 00 00   type ...........
1C30:0320  64 61 74 65 0D 00 00 00-00 00 00 00 00 00 00 00   date............
1C30:0330  74 69 6D 65 0D 00 00 00-00 00 00 00 00 00 00 00   time............
-

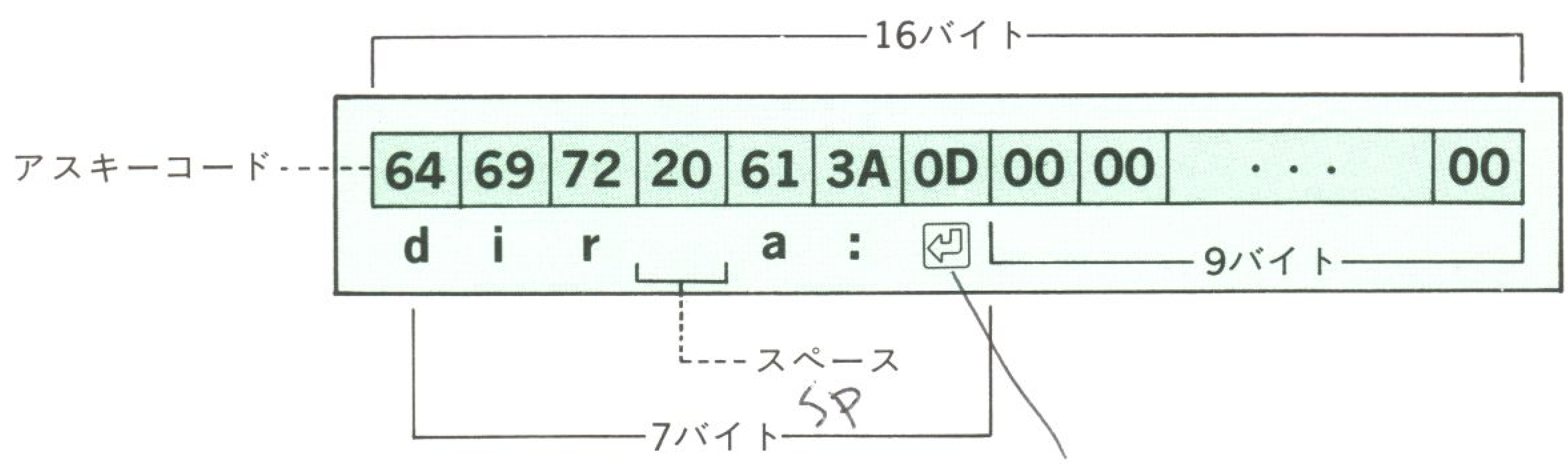
-Q

A>
A>
```

1つのキーの内容に16バイトが使われている

dir a：のところを見てください。1つのファンクションキーの内容は、次のように16バイトが使われています。

■ファンクションキーの格納状態



これにより、キーの内容を設定するには、その内容の文字列を入れて、残りには00を書き込んでおけばよいことがわかります。

先程の「TYPE　」の例では、4バイト＋空白1バイトで5バイトが必要です。そして、残りが16バイト－5バイトで11バイトを00にしておけばOKです。

●DBは、Define　Byte(デファイン・バイト)で1バイトをメモリ中に確保するアセンブラに対する命令。

```
BUFF DB 'TYPE '
     DB 11 DUP (0)
```

DB 11 DUP　(0) というのは、アセンブラの疑似命令で、0を11個メモリのなかに格納するものです。DUPとは、DUPlicate (重複させる、複写する) の略で、その前に書かれている数字の個数だけ ( ) 内のデータを格納する領域を確保するという機能を果たします。

```
DB 10 DUP (1)
```

とすると、1を10個(10バイト分)メモリに格納することになります。

2·4 8086の基本命令を覚える

■繰り返し処理をする（INC・DEC・JNZ）

先程、1個のファンクションキーを設定する方法はわかりました。しかし、設定するキーは10個あります。AXに1, 2, 3, . . . , 10と入れて、10回同じファンクション・コールを実行するのは、無茶というものです。

コンピュータは、繰り返し処理が得意なのですから、10回繰り返しの処理をプログラミングしてみましょう。簡単な例として、0から9までの数を画面に表示するプログラムを作ります。これをアセンブリ言語的な考え方で、BASIC言語で書いたのが、次のプログラムです。

■BASICでの繰り返し処理（アセンブリ言語風）

```
1   ' save "REP.BAS"
100 '---------------------
110 ' Display from 0 to 9
120 '---------------------
130 DL=0
140 BX=10
150 '----------------------
160 *REPEAT
170   PRINT DL;
180   DL=DL+1
190   BX=BX-1
200   IF BX<>0 THEN *REPEAT
210 '----------------------
220 END
```

● 実行結果

```
run ⏎
 0  1  2  3  4  5  6  7  8  9
Ok
```

これと同じ機能をアセンブリ言語で書いたのが、次のプログラムです（数は、文字として表示している点が違いますが）。

■アセンブリ言語での繰り返し処理(一般的)

```
;-----------------------------------
;  REPEAT.ASM
; Display from 0 to 9
; for Tests
;-----------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG   100H
;-----------------------------------
START:
        MOV AH,2          ; Display CHR
        MOV DL,'0'        ; DL=0
        MOV BX,10         ; BX=COUNTER
;-----------------------------------
REPEAT:
        INT 21H           ; Display DL
        INC DL            ; DL=DL+1
        DEC BX            ; BX=BX-1
        JNZ REPEAT        ; IF BX<>0 THEN
                          ; GOTO REPEAT
;-----------------------------------
        MOV AH,0
        INT 21H           ; Return to DOS
;-----------------------------------
CODE    ENDS
        END START
```

```
A>REP ⏎
0123456789
A>
```

プログラムの説明は、次のとおりです。

<MS-DOS のファンクション・コールで1文字表示の準備>

MOV AH, 2 ;1文字表示

MOV DL, '0' ;文字として 0 を入れている

<繰り返し回数を計るカウンタとして BX を使用>

MOV BX, 10 ;10 回を設定

<DL の内容を表示>

INT 21 H

< DL の内容に 1 を足す>

INC DL

INC は、INCrement（増加）のことで、1 を足す 8086 の命令。

● INC は、インクと読み、増加なので＋1 すると覚える。

<BX の内容から 1 を引く>

DEC BX

DEC は、DECrement（減少）のことで、1 を引く命令。

● DEC は、デックと読み、減少なので－1 すると覚える。

<BX が 0 でなければ、REPEAT にジャンプする>

JNZ REPEAT

JNZ は、Jump on Not Zero のことで、0でなければジャンプする8086 の命令。

● JNZ は、ジャンプ・ノット・ゼロと読み、0 でなければジャンプすると覚える。

<MS-DOS のコマンド待ちの状態に戻す>

MOV AH, 0
INT 21 H

AH に 0 を入れて、 INT 21H を実行すると MS-DOS に戻る。

このようにアセンブリ言語では、繰り返し処理は、次のようにする方法が一般的です。

- 繰り返す回数を、あるレジスタに入れる。
- 1 回処理が済むたびに、そのレジスタから 1 を引く。
- それが 0 になったかどうかで、処理の終了を判断する。

```
        MOV BX, 10  ；繰り返す回数
REPEAT：

        繰り返し処理
            ・
            ・
        DEC BX      ；1を引く
        JNZ REPEAT  ；0かどうかで判断
```

●計算の結果が0になれば、ゼロフラグが立つので、JNZはこれをみて0かどうかを判断している。

この例で、BXレジスタを使ったのは、ほかのところでBXが使われていなかったからです。他に使われていないSIやDIなどを使うこともできます。

これを利用して作ったのが、MS-DOSのコマンドを設定するプログラムです。

■MS-DOSコマンド設定プログラム

```
;--------------------------------
; FCOM.ASM
; Function Keys Change
; for MS-DOS commands
;--------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG    100H
;--------------------------------------
START:
        MOV DX,OFFSET BUFF ; Fkeys Addr
        MOV AX,1           ; Fkey Write
        MOV BX,10          ; BX=Counter
REPEAT:
        MOV CL,0DH         ; Fkeys Write
        INT 0DCH           ;
        INC AX             ; AX=AX+1
        ADD DX,16          ; DX=DX+16
        DEC BX             ; BX=BX-1
        JNZ REPEAT         ; IF BX<>0
                           ; then REPEAT
;--------------------------------------
        MOV AH,0
        INT 21H            ; Return to DOS
```

ファンシーキーの内容を1から10まで設定

```
;-----------------------------------------
BUFF    DB 'DIR A:/W',13       ; f・1
        DB 7 DUP (0)

        DB 'DIR B:/W',13       ; f・2
        DB 7 DUP (0)

        DB 'COPY '             ; f・3
        DB 11 DUP (0)

        DB 'DEL '              ; f・4
        DB 12 DUP (0)

        DB 'REN '              ; f・5
        DB 12 DUP (0)

        DB 'FORMAT '           ; f・6
        DB 9 DUP (0)

        DB 'DISKCOPY '         ; f・7
        DB 7 DUP (0)

        DB 'TYPE '             ; f・8
        DB 11 DUP (0)

        DB 'CHDIR '            ; f・9
        DB  10 DUP (0)

        DB 'CLS',13            ; f・10
        DB 12 DUP (0)

CODE    ENDS
        END START
```

設定するファンクションキーの内容

・INC AX でファンクションキーを設定する番号を1つずつ増やしています。
・ADD DX, 16 でファンクションキー格納アドレスを**16バイト**おきに飛ばしています。
・ADD は、その名のとおり加える命令で、DX＝DX＋16 ということになります。

●ADD は、アッドと読み、加えるということなので、足し算をすると覚える。

■8086 特有の繰り返し命令（LOOP）

8086 では、繰り返し処理を効率よくプログラミングするため専用の繰り返し命令 LOOP があります。それは、次のように使います。

●LOOP は、ループと読み、もともとの意味は「環や輪」で「同じところをぐるぐるまわる状態」を指す。

```
REPEAT : MOV CX, 10   ;繰り返す回数

          繰り返し処理
               ・
               ・
          LOOP REPEAT ; CXから1を引いて0かど
                        うかを判断する
```

これは、次のようにしています。

・繰り返しの回数は、CX レジスタに入れる。
・LOOP 命令を実行する。

LOOP(ループ)は、その名のとおり、ループする命令で、これ1つで、

・CX から 1 を引く。
・CX が 0 でなかったら、指定の箇所にジャンプする。
・CXが0でなかったら、LOOPの次の命令に進む。

という機能があります。つまり、LOOP REPEAT は、

```
DEC CX
JNZ REPEAT
```

いう命令を実行することと同じです。

これは、BASIC 言語では FOR-NEXT に近い処理といえましょうが、同じとはいえません。

■BASICでの繰り返し処理(LOOP風)

```
1   ' save "REP2.BAS"
100 '--------------------
110 ' Display from 0 to 9
120 '--------------------
130 DL=0
140 CX=10
150 '---------------------
160 FOR I=CX TO 1 STEP -1
170   PRINT DL;
180   DL=DL+1
190 NEXT
200 '---------------------
210 END
```

● 実行結果

```
run⏎
 0  1  2  3  4  5  6  7  8  9
Ok
```

CX と LOOP を使ったプログラムは、次のようになります。



■LOOPを使った繰り返し処理

```
;----------------------------------
;   REP2.ASM
;  Display from 0 to 9
;  for Tests
;----------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG    100H
;----------------------------------
START:
        MOV AH,2          ; Display CHR
        MOV DL,'0'        ; DL=0
        MOV CX,10         ; CX=COUNTER
;----------------------------------
REPEAT:
        INT 21H           ; Display DL
        INC DL            ; DL=DL+1
        LOOP REPEAT       ; CX=CX-1
                          ; IF CX<>0 THEN
                          ; GOTO REPEAT
;----------------------------------
        MOV AH,0
        INT 21H           ; Return to DOS
;----------------------------------
CODE    ENDS
        END START
```

```
A>REP2 ⏎
0123456789
A>
```

そして、ファンクションキーの内容を初期設定に戻すプログラムは、次のようになります。ファンクションキー格納のしかたは、P149を参考にしています。

■初期設定に戻すプログラム(CXとLOOPを使用)

```
;---------------------------------
; FINIT.ASM
; Function Keys Initialize
; for MS-DOS commands
;---------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG    100H
;--------------------------------------
START:
        MOV DX,OFFSET BUFF ; Fkeys Addr
        MOV CX,10          ; CX=Conter
        MOV AX,1           ; Fkey Write
REPEAT: PUSH CX            ; Save CX
        MOV CL,0DH         ; Fkeys Read
        INT 0DCH           ; and Write
        INC AX             ; AX=AX+1
        ADD DX,16          ; DX=DX+16
        POP CX             ; Restore CX
        LOOP REPEAT        ; IF CX<>0
                           ; then REPEAT
;---------------------------------------
        MOV AH,0
        INT 21H            ; Return to DOS
;---------------------------------------
BUFF    DB 0FEH,' C1   ',1BH,53H ; f・1
        DB 8 DUP (0)

        DB 0FEH,' CU   ',1BH,54H ; f・2
        DB 8 DUP (0)

        DB 0FEH,' CA   ',1BH,55H ; f・3
        DB 8 DUP (0)

        DB 0FEH,' S1   ',1BH,56H ; f・4
        DB 8 DUP (0)

        DB 0FEH,' SU   ',1BH,57H ; f・5
        DB 8 DUP (0)

        DB 0FEH,'VOID  ',1BH,45H ; f・6
        DB 8 DUP (0)

        DB 0FEH,'NWL   ',1BH,4AH ; f・7
        DB 8 DUP (0)

        DB 0FEH,'INS   ',1BH,50H ; f・8
        DB 8 DUP (0)

        DB 0FEH,'REP   ',1BH,51H ; f・9
        DB  8 DUP (0)

        DB 0FEH,' ^Z   ',1BH,5AH ; f・10
        DB 8 DUP (0)

CODE    ENDS
        END START
```

要注意

ファンクションキー
1〜10を設定

初期設定の内容

■レジスタの内容を保存する（PUSH・POP）

新しい命令 PUSH・POP に注意してください。

```
REAPET : PUSH CX
         MOV CL, 0DH
           ・
           ・
           ・
         POP CX
         LOOP REPEAT
```

これは、

PUSH CX

で CX の内容をメモリ（スタックと呼ばれる専用の領域）に退避します。このとき CX の内容は変化せず、同じ内容がメモリに格納されます。そして、

POP CX

でメモリに退避していた CX の内容を CX に戻します。

これは、繰り返し処理のなかで CL レジスタを使用しており、CX の内容が変わるのを防ぐためにしていることです。CX は、CL と CH をあわせたものですから、

MOV CX, 10

とすると、

CX は、000AH

となり、

CL は、0AH
CH は、00H

●**PUSH は、**プッシュと読み、内容をやわらかいところに押して入れる状態を指す。

●**スタックは、**PUSH された内容を保持したり、サブルーチン・コールのあとの戻り先を記録したりする重要なメモリ部分。

●POP は、ポップと読み、内容をポンと取り出す状態を指す。

となることになります。
次に

MOV CL, 0DH

とすると、

CL は、0DH

となり、CH は 00H のままですから、結果として

CX は、000DH

となります。CX にカウンタとして 10 の値を入れていたのが、別な値になってしまいます。これでは、正常に繰り返し処理が行われなくなります。

そこで、PUSH CX, POP CX として、CX の値が変わらないようなテクニックを用いているのです。PUSH・POP は、このように一般的にあるレジスタの内容を一時保存しておいて、のちにもとの内容に戻すときに使います。なぜ、こういうことをするのかといいますと、使えるレジスタの数に限りがあるからです。BASIC 言語のように、ユーザーが勝手に必要なだけ変数を使うことができないためです。

●あるレジスタの値を変えたくない場合は、PUSH・POP を使う。

■サブルーチン・コール（CALL・RET）

繰り返し処理の部分は、2 つにわけて、メインルーチンとサブルーチンにすることができます。ここで、サブルーチン・コールの命令を学ぶために、FCOM. ASM と FINIT. ASM をサブルーチン・コールをするように書き換えてみましょう。

アセンブリ言語でのサブルーチン・コールは、BASIC 言語の GOSUB-RETURN とよく似ています。ほとんど同じといっていいでしょう。CALL と RET を使います。

●CALL は、コールと読み、その名のとおり、サブルーチンをコールする(呼び出す)。

```
        CALL  サブルーチン・ラベル ；GOSUB
              ・
              ・
サブルーチン・ラベル：
              ・
          サブルーチン
              ・
        RET                        ；RETURN
```

●RETは、レットと読み、RETURNの略でサブルーチンからリターンする(戻る)。

これを使って書き換えたのが、次のプログラムです。キーの内容は、8086のニーモニックにしています。ソース・プログラムを書くときに便利でしょう。繰り返しの部分からサブルーチンをコールするようにすると、繰り返し処理がはっきりします。特に、PUSH・POPの箇所がわかりやすくなります。

■サブルーチン・コール(一般的な繰り返し)

```
;--------------------------------
; FASM.ASM
; Function Keys Change
; for Macro Assembler
;--------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG    100H
;--------------------------------------
START:
        MOV DX,OFFSET BUFF  ; Fkeys Addr
        MOV AX,1            ; Fkey Write
        MOV BX,10           ; BX=Counter
;--------------------------------------
REPEAT:
        CALL WRITE          ; GOSUB *WRITE ———— サブルーチン・コール
        DEC BX              ; BX=BX-1
        JNZ REPEAT          ; IF BX<>0
                            ; then REPEAT
;--------------------------------------
 DOS:
        MOV AH,0
        INT 21H           ; Return to DOS
;======================================
WRITE:
        MOV CL,0DH          ; Fkeys Read
        INT 0DCH            ; and Write
        INC AX              ; AX=AX+1
        ADD DX,16           ; DX=DX+16
        RET                 ; RETURN ———— メインルーチンに戻る
;======================================
BUFF    DB 'MOV  '            ; f・1
        DB 12 DUP (0)

        DB 'INT 21H'          ; f・2
        DB 9 DUP (0)

        DB 'INC  '            ; f・3
        DB 12 DUP (0)

        DB 'DEC  '            ; f・4
        DB 12 DUP (0)             ニーモニックを設定

        DB 'JE  '             ; f・5
        DB 13 DUP (0)

        DB 'JNE  '            ; f・6
        DB 12 DUP (0)

        DB 'CALL  '           ; f・7
        DB 11 DUP (0)

        DB 'RET  '            ; f・8
        DB 12 DUP (0)

        DB 'PUSH  '           ; f・9
        DB 11 DUP (0)

        DB 'POP  '            ; f・10
        DB 12 DUP (0)

CODE    ENDS
        END START
```

■サブルーチン・コール(LOOPとPUSH・POPを使用)

```
;--------------------------------
; FASM2.ASM
; Function Keys Change
; for Macro Assembler
;--------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG    100H
;--------------------------------------
START:
        MOV DX,OFFSET BUFF  ; Fkeys Addr
        MOV AX,1            ; Fkey Write
        MOV CX,10           ; CX=Counter
;--------------------------------------
REPEAT:
        PUSH CX             ; Save CX
        CALL WRITE          ; GOSUB *WRITE      サブルーチン・コール
        POP CX              ; Restore CX
        LOOP REPEAT         ; IF CX<>0
                            ; then REPEAT
;--------------------------------------
 DOS:
        MOV AH,0
        INT 21H           ; Return to DOS
;======================================
WRITE:
        MOV CL,0DH          ; Fkeys Read
        INT 0DCH            ; and Write
        INC AX              ; AX=AX+1
        ADD DX,16           ; DX=DX+16
        RET                 ; RETURN      メインルーチンに戻る
;======================================
BUFF    DB 'MOV '             ; f・1
        DB 12 DUP (0)

        DB 'INT 21H'          ; f・2
        DB 9 DUP (0)

        DB 'INC '             ; f・3
        DB 12 DUP (0)

        DB 'DEC '             ; f・4
        DB 12 DUP (0)

        DB 'JE '              ; f・5
        DB 13 DUP (0)

        DB 'JNE '             ; f・6
        DB 12 DUP (0)

        DB 'CALL '            ; f・7
        DB 11 DUP (0)

        DB 'RET '             ; f・8
        DB 12 DUP (0)

        DB 'PUSH '            ; f・9
        DB 11 DUP (0)

        DB 'POP '             ; f・10
        DB 12 DUP (0)

CODE    ENDS
        END START
```

2・5 ファンクションキーの内容入れ替えコマンドを作る

MS-DOSでファンクションキーを使うには、

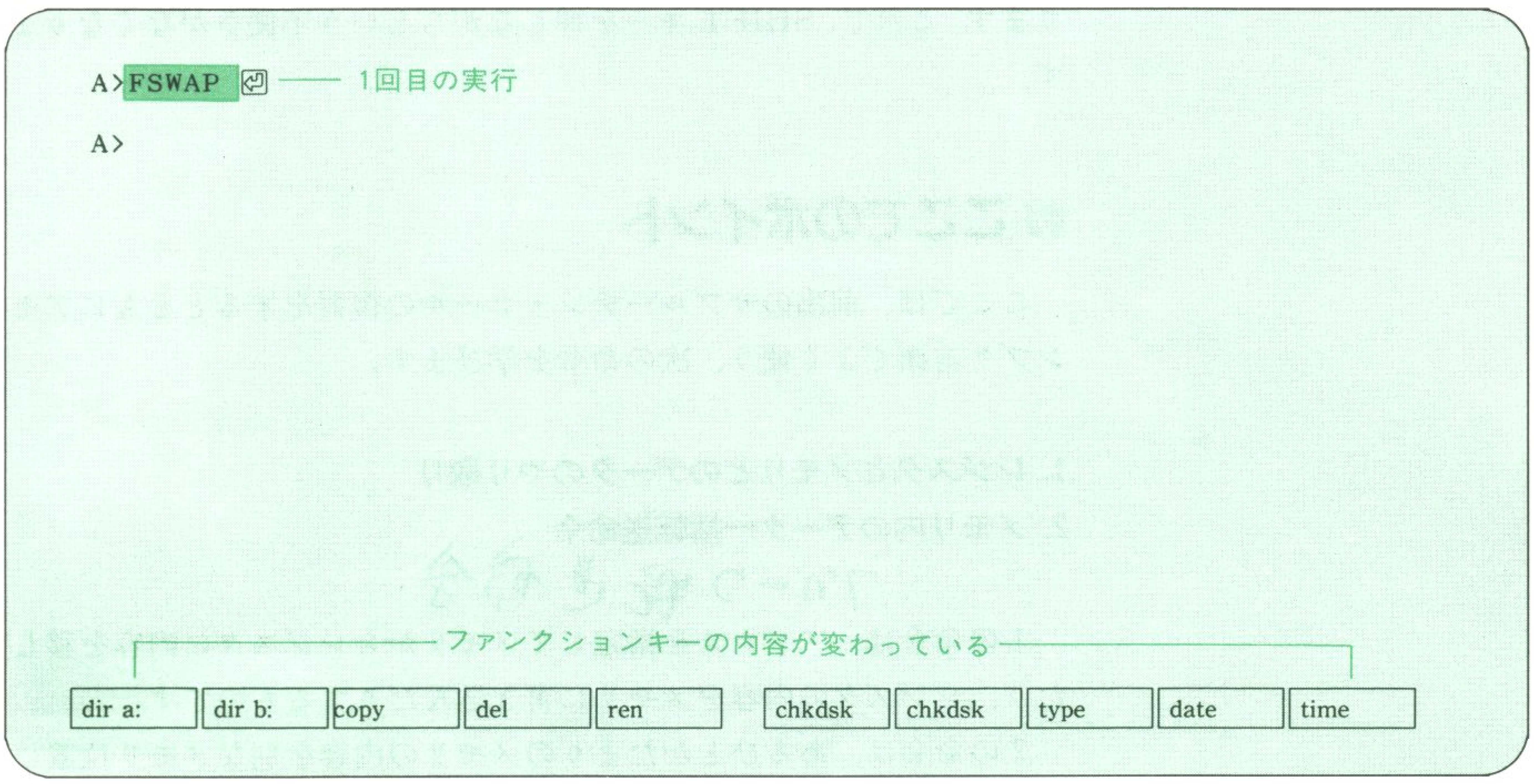
f・1 〜f・10のときは、そのまま各キーを押す。
f・11〜f・20のときは、SHIFTキーを押しながら各キーを押す。

ということになっています。

これでは、f・11〜f・20のキーが使いにくいので、使う内容に応じて、f・1 〜 f・10とf・11〜 f・20との内容を入れ替えるプログラムを作りましょう。

まず、実行例を次に挙げます(FSWAPは、Function Keys Swap：交換という意味で付けたファイル名)。

■ファンクションキーの内容を入れ替える



```
A>FSWAP

A>FSWAP ⏎ ── 2回目の実行

A>



        もとの内容に戻っている
 C1   CU   CA   S1   SU    VOID  NWL  INS  REP  ^Z
```

FSWAPを実行するたびに、ファンクションキーの内容が入れ替わります。これで、SHIFTキーを押しながらという不便さがなくなります。

■ ここでのポイント

ここでは、前出のサブルーチン・コールの復習をするとともにアセンブリ言語でよく使う、次の命令を学びます。

1. レジスタとメモリとのデータのやり取り
2. メモリ内のデーター括転送命令

ブロック転送命令.

1の命令は、アドレスを指定してメモリからレジスタに内容を移したり、レジスタの内容をメモリに書き込んだりするものです。

2の命令は、あるひとかたまりのメモリの内容を別なメモリ位置に転送するもので、ブロック（ひとかたまりの意）転送命令といいます。

■ メモリの内容を入れ替えるには（[SI]，[DI]）

ファンクションキーの内容を入れ替えるには、メモリに読み込んだキーの内容を入れ替えればOKです。メモリAとメモリBの内容を入れ替えるには、通常、次の作業を行います。

- メモリ A の内容をあるレジスタに入れる。
- メモリ B の内容を別なレジスタに入れる。
- あるレジスタ（メモリ A）の内容をメモリ B に入れる。
- 別なレジスタ（メモリ B）の内容をメモリ A に入れる。

メモリ（アドレス 0）の内容 1 バイトをレジスタ AL に入れるには、普通、次のようにします。

```
MOV SI, 0
MOV AL, [SI]
```

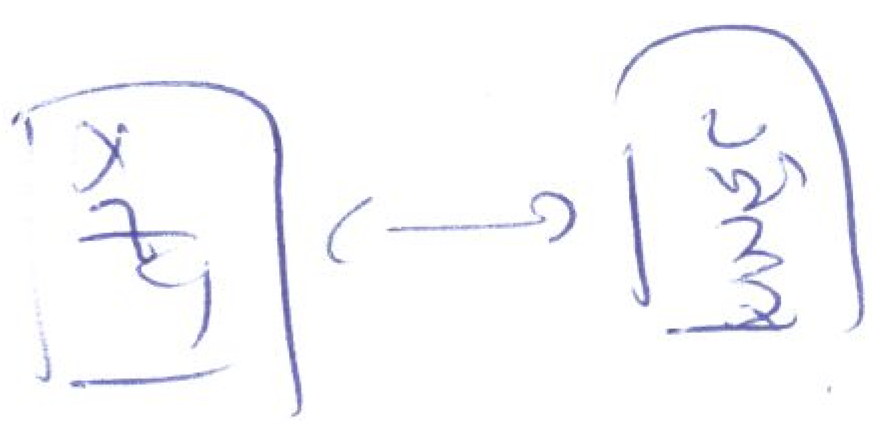
● MOV AL, [SI] は、SI が指すアドレスの内容を AL に入れる。SI＝0 で、アドレス 0 の内容が 1 であれば、AL＝1 となる。

この意味は、次のとおりです。

- アドレス指定を行うレジスタの 1 つ SI に、アドレス 0 を入れる。
- [SI] とレジスタを大カッコでくくると、SI が示すアドレスのメモリ内容を表わす。
- SI が示すアドレスのメモリ内容を AL に入れる。

ここで、レジスタを大カッコでくくると、その値ではなく、その値（アドレス）が示すメモリの内容となると覚えておいてください。

同様に、メモリ（アドレス 0）にレジスタ AL の内容を入れるには、普通、次のようにします。



```
MOV SI, 0
MOV  [SI], AL
```

● MOV [SI]、AL は、SI が指すアドレスに AL の内容を入れる。SI＝0 で、AL の内容が 1 であれば、アドレス 0 の内容は、1 となる。

アドレス指定を行うもう 1 つのレジスタ DI についても、同じです。

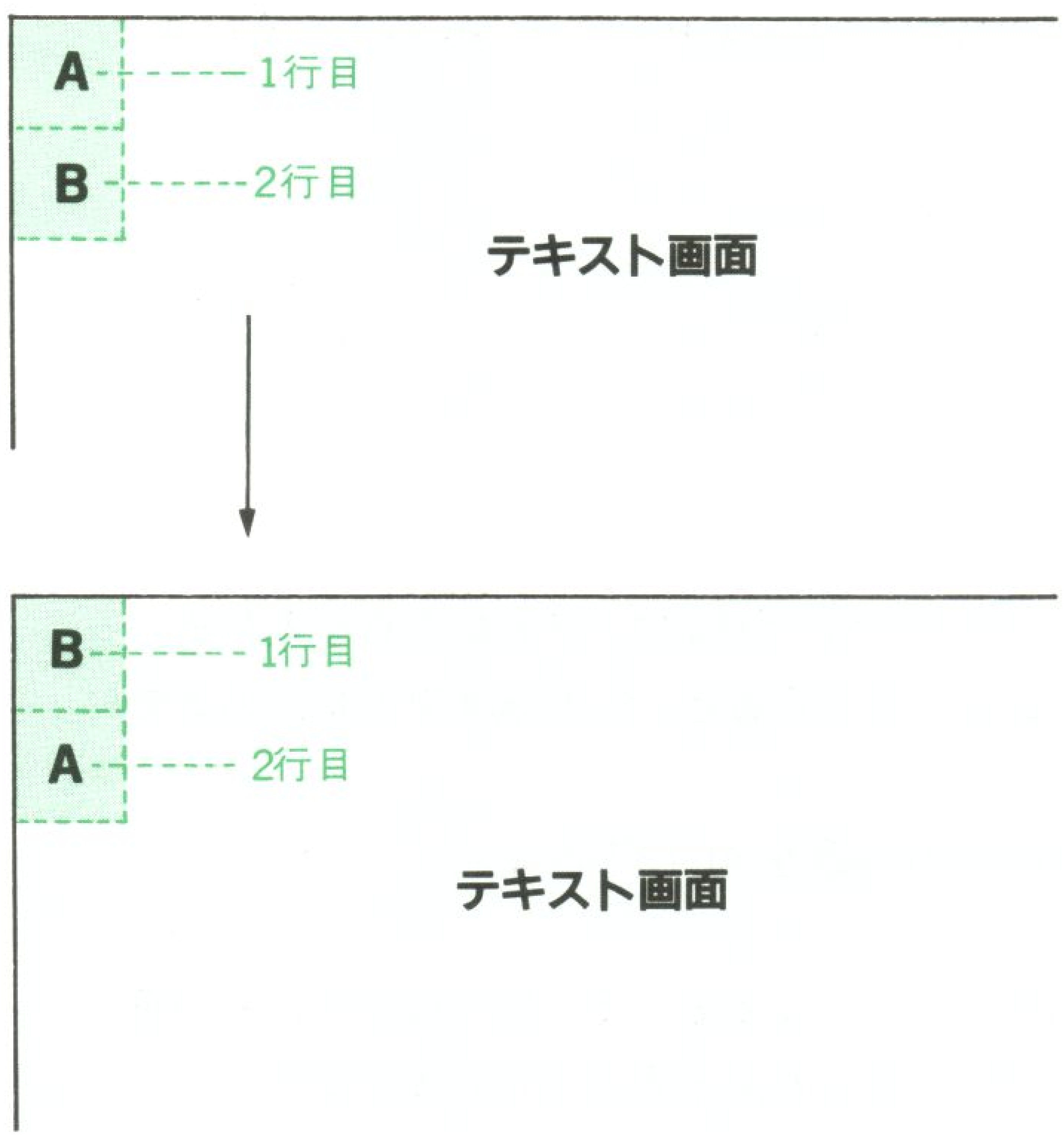
```
MOV DI, 0
MOV AL,[DI]
MOV  [DI], AL
```

画面表示で練習する

レジスタとメモリとの読み書きの具体例を示します。これで実際の働きを体得してください。例として、次のようなプログラムを取り上げます。

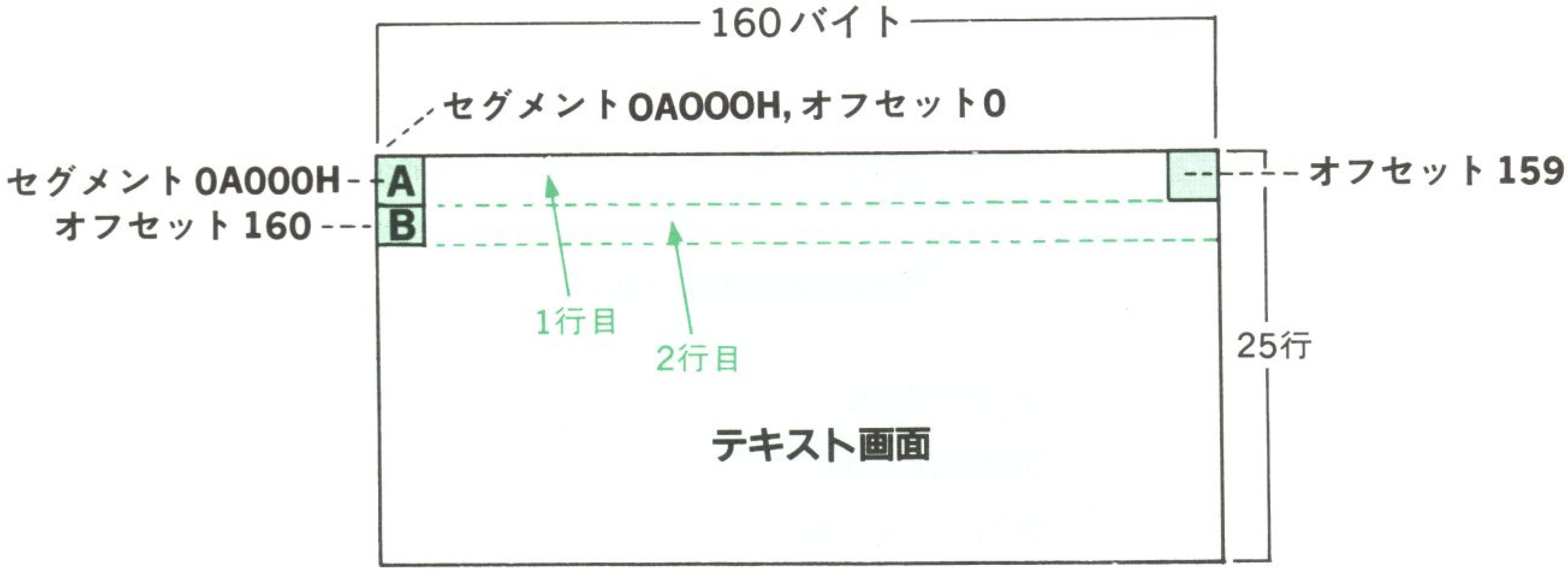
・テキスト画面の１行目に「A」と表示する。
・テキスト画面の２行目に「B」と表示する。
・「A」「B」の表示を入れ替える。

■表示「A」「B」の入れ替え



98 のテキスト画面は、次のようになっており、半角の英数字で 80 字×25 行表示できます。これだけなら 1 行 80 バイトですむのですが、実際には 1 行 160 バイト分が用意されています。これは漢字表示のためです(くわしくは述べませんが、2 バイトの漢字 1 文字を表示するのに、VRAM を 4 バイト必要とするのです)。

■98のテキスト画面



テキスト画面のセグメント・アドレスとオフセット・アドレスは、次のとおりです。

- セグメントは、0A000H
- 1行目の1文字目のオフセットは、0
- 2行目の1文字目のオフセットは、160

●英数字であれば、そのアスキーコードをテキスト画面用メモリに書き込めば、それに対応した文字が画面に表示される。

これらのことがわかれば、次のように課題のプログラムが作れます。

<テキスト画面の1行目に「A」と表示する>

```
MOV AX, 0A000H
MOV DS, AX
MOV SI, 0
MOV AL, 'A'
MOV  [SI], AL
```

CS, DS, SS, ES セグメント指定.
SP, BP, SI, DI アドレス指定.

最初の2行で、

```
MOV AX, A000H
MOV DS, AX
```

としてDSに0A000Hを入れています。DSにアドレスを入れるには、通常AXを介します。これは、次のように直接DSに値を入れることができないためです。

× **MOV DS, 0A000H**

セグメント・レジスタでは、直接、値を入れることができません。なぜ、そうなのかと問うより、8086の設計がそうなっていると理解して、覚えた方がよいでしょう。これは、言葉を覚えることと同じです。

<テキスト画面の2行目に「B」と表示する>

MOV DI, 160
MOV AL, 'B'
MOV [DI], AL

<「A」「B」の表示を入れ替える>

MOV AL, [SI]
MOV AH, [DI]
MOV [SI], AL
MOV [DI], AH

以上をソース・プログラムにしたのが、次のものです。

■表示「A」「B」入れ替えプログラム

```
;---------------------------------
; MSWAP.ASM
; Memory Contents Swap
;---------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG    100H
;---------------------------------------
START:
        MOV AX,0A000H                                     DSにテキスト画面の
        MOV DS,AX         ; DS <= 0A000H  ———————  セグメントを入れる
        MOV SI,0  ——————————————————————————  SIにオフセット0を入れる
        MOV AL,'A'
        MOV [SI],AL       ; 0A000H:0 <='A'  ————  「A」を表示
        MOV DI,160
        MOV AL,'B'
        MOV [DI],AL       ; 0A000H:160 <='B'  ——  「B」を表示
;---------------------------------------
INPUT:
        MOV AH,8          ; AL$=INPUT$(1)  ————  キー入力を待つ
        INT 21H
;---------------------------------------
```

```
SWAP:
        MOV AL,[SI]      ; AL <= 'A'           ―― 「A」をとりだす ┐
        MOV AH,[DI]      ; AH <= 'B'           ―― 「B」をとりだす │ 入れ替え
        MOV [SI],AH      ; 0A000H:0 <='B'      ―― 「B」を入れる   │
        MOV [DI],AL      ; 0A000H:160<='A'     ―― 「A」を入れる   ┘
;---------------------------------------
DOS:
        MOV AH,0
        INT 21H            ; Return to DOS   ―― MS-DOSに戻る
;---------------------------------------
CODE    ENDS
        END START
```

なお、「A」「B」を表示したあと、何かキーが押されたら表示を入れ替えるようにしています。これには、AH に 8 を入れ、INT 21H を実行しています。これは、AH が 1 の場合とよく似ていて、キーボードからの入力を待ち、1文字入力されるとその文字を AL に格納してもどります（ここでは、AL に格納された文字は使っていませんが）。ひとつだけ違うのは、入力された文字を画面に表示しないことです。次に、実行例を挙げます。

● MOV AH,8
INT 21H
は、1文字入力のファンクション・コールで入力された文字は画面にでない。

■「A」「B」入れ替えの実行例

```
A ――――― 「A」が表示
B> ――――― 「B」が表示
A>
A>
A>MSWAP ⏎ ――――― 「A」「B」交換プログラムの実行

        ここでなにかキーを押す
```

```
B  ――――― 表示が入れ替わっている
A> ―――――
A>
A>
A>MSWAP

A>
```

■ファンクションキーの内容入れ替えプログラム

さて、いよいよファンクションキーの内容を入れ替えるプログラムを作りましょう。

ファンクションキーの読み書きの方法は、次のとおりです。

DX ←内容を読み書きする先頭アドレス
CL ← 0CH（読み込み）
　　← 0DH（書き込み）
AX ← 0（すべてのキー（[f･1]～[f･20]）の読み書き）
INT 0DCH（読み書きの実行）

これで、読み書きで値が異なるのはDX, CLの値だけで、AXの値とINT 0 DCHは変わりませんので、変わらない部分を次のようにサブルーチンにします。

```
FKEYS：MOV AX, 0
       INT 0DCH
       RET
```

サブルーチン・コールのしかたは、次のようになります。

<読み込み>

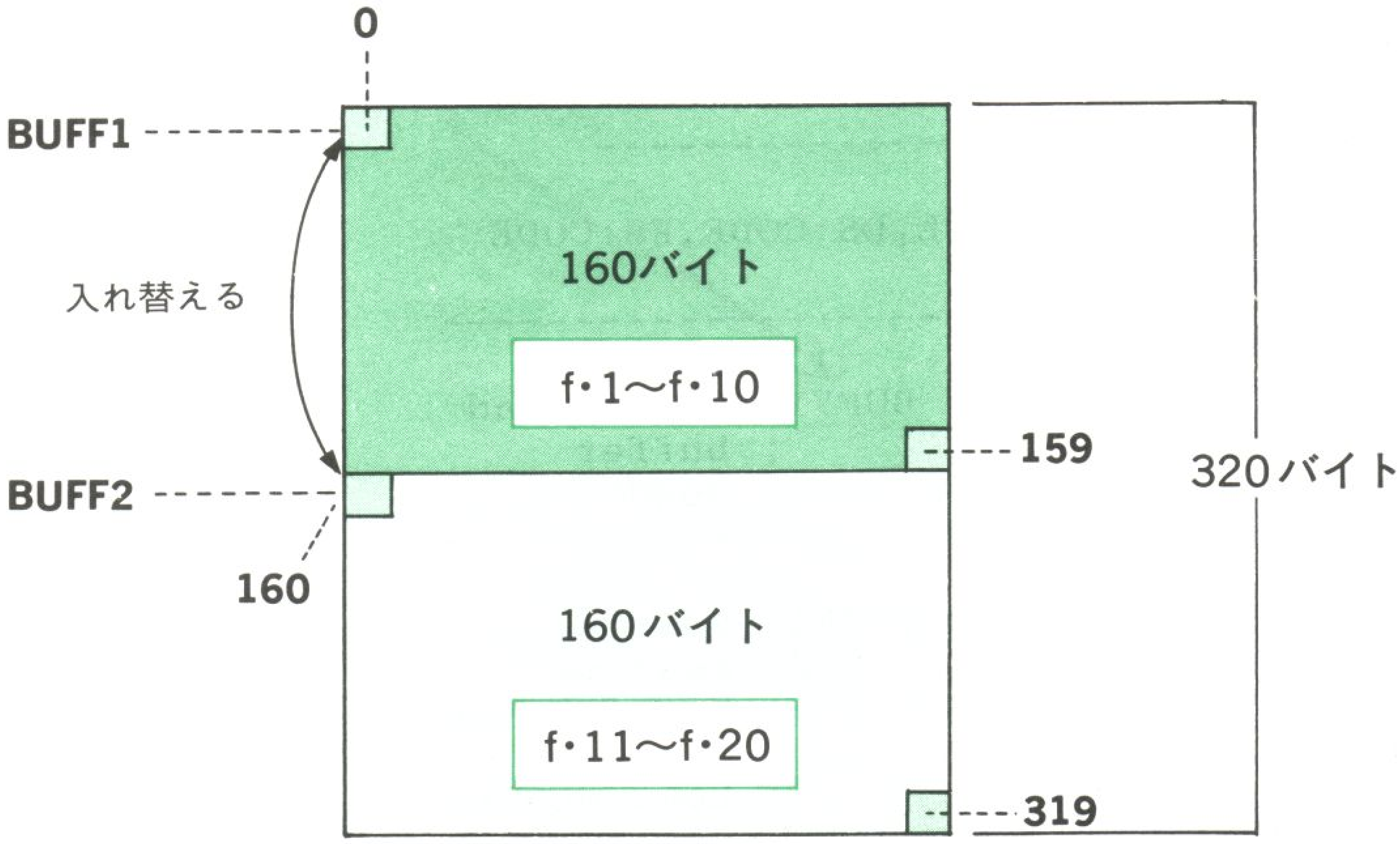
```
MOV DX, 読み込む先頭アドレス
MOV CL, 0CH ；読み込み
CALL FKEYS  ；コール
```

<書き込み>

```
MOV DX, 書き込む先頭アドレス
MOV CL, 0DH ；書き込み
CALL FKEYS  ；コール
```

読み込みは、次のように行います。

■ファンクションキー内容の読み込み



そして、BUFF1とBUFF2が示すアドレスをもとにメモリ内容を入れ替えます。入れ替える内容が160バイトありますので、1文字入れ替えを160回繰り返すことになります。これは、本章の2−4で学んだLOOP命令を使い、次のようにします。

```
        MOV CX, 160
REPEAT: MOV AL, [SI]
        MOV AH, [DI]
        MOV  [SI], AH
        MOV  [DI], AL
        INC SI
        INC DI
        LOOP REPEAT
```

●LOOPを使って繰り返し処理を160回行っている。

INC SI, INC DIで、アドレスを1つずつ増やして、メモリの内容を1バイトずつずらしています。こういうふうにアドレス指定のレジスタを使います。

次に、全体のプログラムを挙げます。

■ファンクションキー入れ替えプログラム(1バイト単位)

```
;---------------------------------
; FSWAP.ASM
; Function Keys Swap
; f・1-f・10 <=>f・11-f・20
;---------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG    100H
;---------------------------------------
START:
READ:   MOV DX,OFFSET BUFF1; Fkeys Read
        MOV CL,0CH         ; buffer
        CALL FKEYS         ; Read
;---------------------------------------
SWAP:
        MOV SI,OFFSET BUFF1; Source ad
        MOV DI,OFFSET BUFF2; Dest ad
        MOV CX,160         ; No.of bytes
REPEAT:
        MOV AL,[SI]        ; Load contents
        MOV AH,[DI]
        MOV [SI],AH        ; Swap
        MOV [DI],AL
        INC SI             ; SI=SI+1
        INC DI             ; DI=DI+1
        LOOP REPEAT
;---------------------------------------
WRITE:
        MOV DX,OFFSET BUFF1
        MOV CL,0DH         ; Fkeys
        CALL FKEYS         ; Write
;---------------------------------------
DOS:
        MOV AH,0
        INT 21H            ; Return to DOS
;=======================================
FKEYS:
        MOV AX,0           ; All Fkeys
        INT 0DCH           ; Fkeys Set
        RET                ; RETURN
;=======================================
BUFF1   DB 160  DUP (?)    ; 320 BYTES
BUFF2   DB 160  DUP (?)    ; RESERVED
CODE    ENDS
        END START
```

- START/READ: ファンクションキー読み込み
- MOV SI / MOV DI: 各先頭アドレス・セット
- MOV CX,160: 入れ替えバイト数セット
- REPEAT: 1バイト単位で入れ替え
- WRITE: ファンクションキーの書き込み
- DOS: MS-DOSへ戻る
- FKEYS: ファンクションキー読み書きサブルーチン
- BUFF1: f・1〜f・10のバッファ確保
- BUFF2: f・11〜f・20のバッファ確保

■ 2バイトずつの入れ替え

これまでは、1バイト単位での入れ替えを行いましたが、8086は、16ビットのCPUですから、1度に16ビット(2バイト)の処理ができます。次に、2バイト単位での入れ替えを行ってみましょう。その部分は、次のようになります。

```
           MOV CX, 80
REPEAT : MOV AX, [SI]
           MOV BX, [DI]
           MOV  [SI], BX
           MOV  [DI], AX
           INC SI
           INC SI
           INC DI
           INC DI
           LOOP REPEAT
```

ポイントは、次のとおりです。

- 2バイト単位なので、CXに入れる値は、半分の80でよい。
- 16ビットのレジスタAX, BXを使用している。
- MOV AX, [SI] とすると、

  AL ← [SI]

  AH ← [SI+1]

  となり、1度に2バイトを読むことができる。
- MOV [DI], AX とすると、

  [DI] ← AL

  [DI+1] ← AH

  となり、1度に2バイトを書くことができる。
- [SI+1], [DI+1] は、SI, DI のアドレスに+1したアドレスの内容を表わします。こういうふうに、+−でレジスタの前後のアドレスを指定することができます。
- 2バイト単位なので、INC SI, INC SI と読むアドレスを2つずつ増やしている。
- 同様に、書くアドレスも2つずつ増やしている。

● INC の代わりに
ADD SI,2
ADD DI,2
としてもよい。

2バイトずつの入れ替えプログラム

次に、2バイトずつの入れ替えプログラムを示します。この例では、値にシンボル（ラベル）を使ってみました。

これをCOMファイルにして、実行してみてください。試しに、デバッガを使って実行し、ファンクションキーの内容が入れ替わっているかみてみましょう。次のように操作してください。

■ファンクションキー入れ替えプログラム(2バイト単位)

```
;---------------------------------
; FSWAP2.ASM
; Function Keys Swap 2
; f・1-f・10 <=>f・11-f・20
;---------------------------------
FREAD   EQU 0CH
FWRITE  EQU 0DH
ALLKEYS EQU 0
FSET    EQU 0DCH
NO      EQU 80        ; No. of words:80
;---------------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG    100H
;---------------------------------------
START:
READ:   MOV DX,OFFSET BUFF1; Fkeys Read
        MOV CL,FREAD        ; buffer
        CALL FKEYS          ; Read
;---------------------------------------
SWAP:
        MOV SI,OFFSET BUFF1; Source ad
        MOV DI,OFFSET BUFF2; Dest ad
        MOV CX,NO           ; No.of words
REPEAT:
        MOV AX,[SI]         ; Load contens
        MOV BX,[DI]
        MOV [SI],BX         ; Swap
        MOV [DI],AX
        INC SI              ; SI=SI+2
        INC SI
        INC DI              ; DI=DI+2
        INC DI
        LOOP REPEAT
;---------------------------------------
WRITE:
        MOV DX,OFFSET BUFF1
        MOV CL,FWRITE       ; Fkeys
        CALL FKEYS          ; Write
;---------------------------------------
DOS:
        MOV AH,0
        INT 21H          ; Return to DOS
;=======================================
FKEYS:
        MOV AX,ALLKEYS      ; All Fkeys
        INT FSET            ; Fkeys Set
        RET                 ; RETURN
;=======================================
BUFF1   DB 160  DUP (?)     ; 320 BYTES
BUFF2   DB 160  DUP (?)     ; RESERVED
CODE    ENDS
        END START
```

シンボル定義(FREAD〜NO)

80ワード(160バイト)(NO EQU 80)

入れ替えワード数80をセット(MOV CX,NO)

2バイト単位で入れ替え(REPEAT〜LOOP REPEAT)

■デバッガで実行結果確認

```
A>SYMDEB FSWAP2.COM ⏎
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-U
3952:0100 BA3101          MOV     DX,0131
3952:0103 B10C            MOV     CL,0C        BUFF1のアドレス
3952:0105 E82300          CALL    012B
3952:0108 BE3101          MOV     SI,0131
3952:010B BFD101          MOV     DI,01D1      BUFF2のアドレス
3952:010E B95000          MOV     CX,0050        (160は16進でA0)
3952:0111 8B04            MOV     AX,[SI]
3952:0113 8B1D            MOV     BX,[DI]
-U
3952:0115 891C            MOV     [SI],BX
3952:0117 8905            MOV     [DI],AX
3952:0119 46              INC     SI
3952:011A 46              INC     SI
3952:011B 47              INC     DI
3952:011C 47              INC     DI
3952:011D E2F2            LOOP    0111
3952:011F BA3101          MOV     DX,0131
-
-U
3952:0122 B10D            MOV     CL,0D
3952:0124 E80400          CALL    012B
3952:0127 B400            MOV     AH,00        MS-DOSに戻る
3952:0129 CD21            INT     21
3952:012B B80000          MOV     AX,0000
3952:012E CDDC            INT     DC
3952:0130 C3              RET
3952:0131 0000            ADD     [BX+SI],AL
-
-
-G127 ⏎ ──── MS-DOSに戻る一歩手前まで実行
AX=0000  BX=0000  CX=000D  DX=0131  SP=FFFE  BP=0000  SI=01D1  DI=0271
DS=3952  ES=3952  SS=3952  CS=3952  IP=0127   NV UP EI PL NZ NA PE NC
3952:0127 B400            MOV     AH,00
-D131,26F ⏎  ──── 131H(BUFF1)
3952:0130     64 69 72 20 61 3A 0D-00 00 00 00 00 00 00 00   dir a:.........
3952:0140  00 64 69 72 20 62 3A 0D-00 00 00 00 00 00 00 00  .dir b:.........
3952:0150  00 63 6F 70 79 20 00 00-00 00 00 00 00 00 00 00  .copy ..........
3952:0160  00 64 65 6C 20 00 00 00-00 00 00 00 00 00 00 00  .del ...........
3952:0170  00 72 65 6E 20 00 00 00-00 00 00 00 00 00 00 00  .ren ...........
3952:0180  00 63 68 6B 64 73 6B 20-61 3A 0D 00 00 00 00 00  .chkdsk a:......
3952:0190  00 63 68 6B 64 73 6B 20-62 3A 0D 00 00 00 00 00  .chkdsk b:......
3952:01A0  00 74 79 70 65 20 00 00-00 00 00 00 00 00 00 00  .type ..........
3952:01B0  00 64 61 74 65 0D 00 00-00 00 00 00 00 00 00 00  .date...........
3952:01C0  00 74 69 6D 65 0D 00 00-00 00 00 00 00 00 00 00  .time...........
3952:01D0  00 FE 20 43 31 20 20 1B-53 00 00 00 00 00 00 00  .~ C1   .S.......
3952:01E0  00 FE 20 43 55 20 20 1B-54 00 00 00 00 00 00 00  .~ CU   .T.......
3952:01F0  00 FE 20 43 41 20 20 1B-55 00 00 00 00 00 00 00  .~ CA   .U.......
3952:0200  00 FE 20 53 31 20 20 1B-56 00 00 00 00 00 00 00  .~ S1   .V.......
3952:0210  00 FE 20 53 55 20 20 1B-57 00 00 00 00 00 00 00  .~ SU   .W.......
3952:0220  00 FE 56 4F 49 44 20 1B-45 00 00 00 00 00 00 00  .~VOID .E.......
3952:0230  00 FE 4E 57 4C 20 20 1B-4A 00 00 00 00 00 00 00  .~NWL   .J.......
3952:0240  00 FE 49 4E 53 20 20 1B-50 00 00 00 00 00 00 00  .~INS   .P.......
3952:0250  00 FE 52 45 50 20 20 1B-51 00 00 00 00 00 00 00  .~REP   .Q.......
3952:0260  00 FE 20 5E 5A 20 20 1B-5A 00 00 00 00 00 00 00  .~ ^Z   .Z.......
-Q ⏎
              └── 1D1H(BUFF2)                                 └── 入れ替わっている
                  (160は16進でA0)
A>
```

ブロック転送での入れ替え

前項で行ったメモリの内容入れ替えは、ブロック転送を行うと簡単に、しかも効率よくプログラムができます。

まずは、ブロック転送の効果のほどをお目にかけましょう。

次の例は、テキスト画面に表示された内容をそっくりそのままあるメモリにブロック転送（退避）して、あとで、その内容を復帰させたものです（ファイル名は、TRANSfer（転送）とRESTore（復帰）の省略形を使っています）。

テキスト画面の退避・復帰

● 画面の退避

```
A>DIR ⏎

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

COMMAND  COM    24145  86-10-13   0:00
MASM     EXE    77362  86-10-13   0:00
CREF     EXE    10544  86-10-13   0:00
LINK     EXE    41114  86-10-13   0:00
LIB      EXE    24138  86-10-13   0:00
MAKE     EXE    18675  86-10-13   0:00
MAPSYM   EXE    51904  86-10-13   0:00
SYMDEB   EXE    36538  86-10-13   0:00
YESNO    EXE      801  87-07-22  16:49
VE       EXE    45926  85-03-19  14:21
ASM      BAT       ^C

A>TRANS ⏎ ―― 退避

A>                    この表示内容を退避させる
```

●画面の復帰

```
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>⏎
A>REST⏎  ——復帰
```

ここにDIRの表示を戻す

●復帰した結果

```
A>DIR

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

COMMAND  COM    24145  86-10-13   0:00
MASM     EXE    77362  86-10-13   0:00
CREF     EXE    10544  86-10-13   0:00
LINK     EXE    41114  86-10-13   0:00
LIB      EXE    24138  86-10-13   0:00
MAKE     EXE    18675  86-10-13   0:00
MAPSYM   EXE    51904  86-10-13   0:00
SYMDEB   EXE    36538  86-10-13   0:00
YESNO    EXE      801  87-07-22  16:49
VE       EXE    45926  85-03-19  14:21
ASM      BAT       ^C

A>TRANS   ——退避コマンドの実行のあとまで戻っている

                                もとの表示に戻っている

A>
```

こういうことができると、次のような利用が考えられます。

・よく画面に表示するメッセージは、あらかじめメモリに格納しておくと、簡単に素早く表示できる。

・メニュー表示など頻繁に画面に出す必要があるものも、同じことができる。
・画面表示の結果を一時保存しておき、あとで再表示できる。

これは、ブロック転送の一例ですが、その他いろいろな有効利用が考えられます。

■テキスト画面退避・復帰で練習する

ファンクションキーの入れ替えプログラムを作る前に、テキスト画面退避・復帰プログラムを作って、8086のブロック転送命令をマスターしましょう。

8086のブロック転送命令は、次のように使います。

＜転送元＞

DS ←転送元のセグメント・アドレス
SI ←転送元のオフセット・アドレス

＜転送先＞

ES ←転送先のセグメント・アドレス
DI ←転送先のオフセット・アドレス

＜転送量＞

CX ←転送するバイト数またはワード数

＜転送方向＞

CLD：オフセット・アドレスの小さい方から大きい方へ
STD：オフセット・アドレスの大きい方から小さい方へ

＜ブロック転送命令＞

REP MOVSB：バイト単位の転送
REP MOVSW：ワード単位の転送

●CXの値は、ブロック転送命令実行において、1つずつ減らされる。カウンタとして使われる。

●CLDは、CLear Directionの略で、ブロック転送命令実行においてSI,DIの値を＋の方向に設定する(値を1つずつ増やす)。

●STDは、SeT Directionの略で、ブロック転送命令実行においてSI,DIの値を－の方向に設定する(1つずつ減らす)。

●REP MOVSBは、REPeat MOVe String Byteのことで1バイトずつ一連のメモリ内容(ストリング)を繰り返し動かす。

転送元、転送先、転送量、転送方向を各レジスタに入れて転送の準備をします。そして、ブロック転送命令を実行します。

例えば、

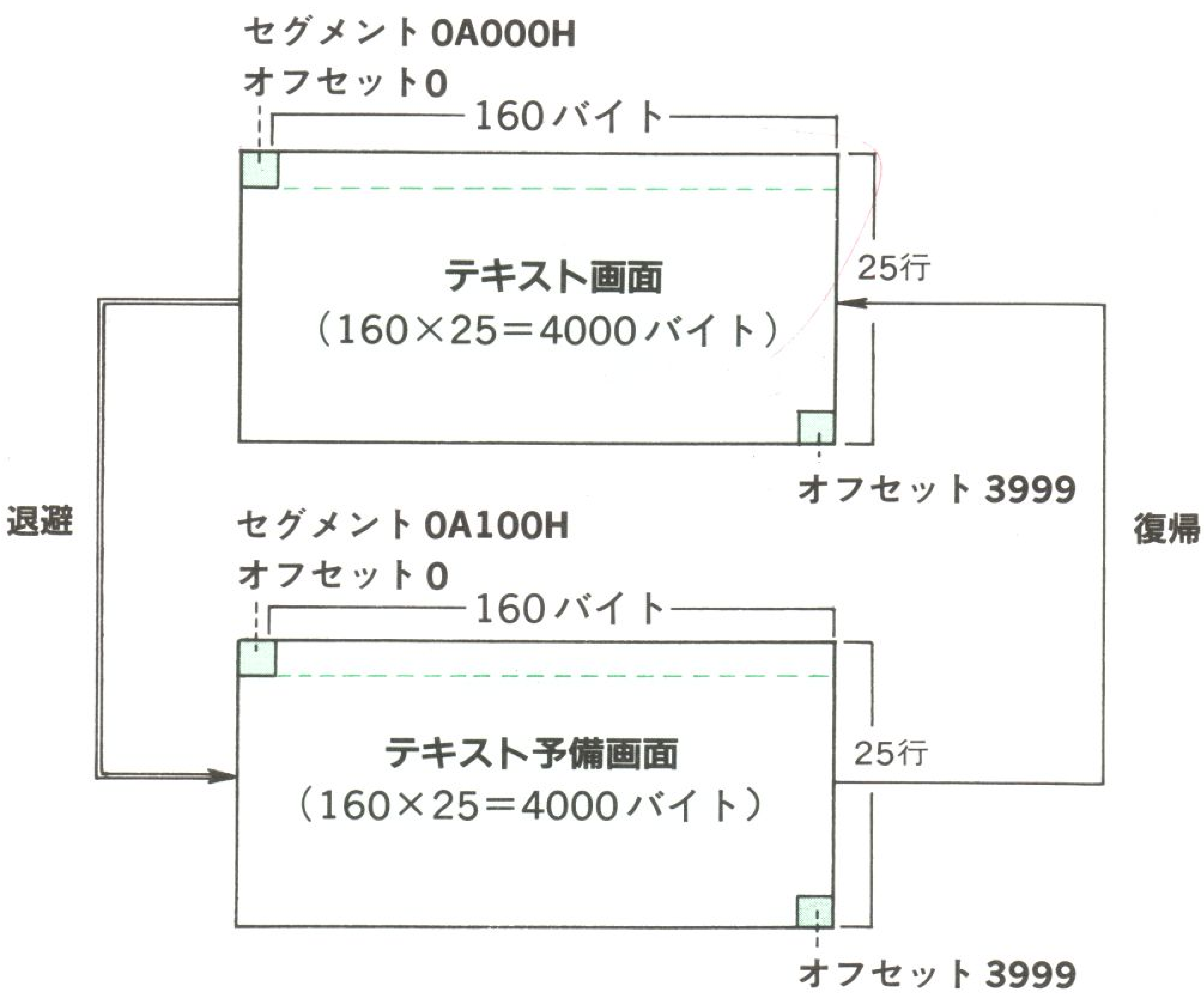
セグメント 0A000H のオフセット 0 から 160 バイト分をセグメント 0A100H のオフセット 0 からに転送するには次のようにします。

●REP MOVSW は、REPeat MOVe String Word のことで 2 バイトずつ一連のメモリ内容(ストリング)を繰り返し動かす。

```
MOV AX, 0A000H
MOV DS, AX
MOV AX, 0A100H
MOV ES, AX
MOV SI, 0
MOV DI, 0
MOV CX, 160
CLD
REP MOVSB
```

こうすると、テキスト画面の 1 行目が退避されます。それは実は、98 のテキスト画面とその予備画面のアドレスが次のようになっているからです。

■98のテキスト画面とテキスト予備画面



テキスト画面は、半角の英数字で80字×25行表示できますが、1行160バイト分が用意されています。そしてまったく同じ構成で、もう1画面、テキスト予備画面があります。

以下、このプログラム例の説明をします。

```
MOV AX, 0A000H
MOV DS, AX
```

DSに0A000Hを入れるには、通常AXを介します。これは、直接DSに値を入れることができないためです。

```
MOV AX, 0A000H
MOV ES, AX
```

同様に、ESに対してもAXを介して値を入れます。

```
MOV SI, 0
MOV DI, 0
```

SIとDIには、それぞれ転送元と転送先のオフセット0を入れています。SIとDIの名前は、こういう機能があるため、

SI：Source Index（元、源を示すもの）
DI：Destination Index（目的地を示すもの）

と付けられているのです。

```
MOV CX, 160
```

CXには、転送するバイト数を入れています。CXは、カウンタ(数をかぞえるもの）としての機能があるため、

CX：Counter Register

と名付けられています。

```
CLD
```

CLD とは、CLear Direction flag ということで、ディレクション（方向）フラグをクリアする（0にする）命令です。こうすると、1バイト転送するごとに、

SI＝SI＋1
DI＝DI＋1

と、それぞれのオフセットの値が自動的に＋1されます。

なお、STD とすると、Set Direction Flag でディレクションフラグがセットされ（1になり）、

SI＝SI－1
DI＝DI－1

という動作になります。

STD は、アドレスの大きいところから小さいところにブロック転送するときに使います。

REP MOVSB

REP とは、REPeat ということで、その次の命令を CX の値が 0 になるまで繰り返すという指示になります。

つぎの命令 MOVSB とは、MOVe String Byte ということで、バイト単位で一連のメモリ（これをストリングという）を転送します。

なお、MOVSW の W とは、Word（ワード：2バイト）のことで、ワード単位で（2バイトずつ）、転送する命令です。

テキスト画面退避プログラム

テキスト画面をすべて退避するには、前のプログラム例で、

MOV CX, 4000

とすればよいことになります。

これは、テキスト画面が、

160 バイト×25 行で 4000 バイト

あるからです。

次に、そのプログラムを挙げます。

■テキスト画面退避プログラム

```
;-------------------------------
; TRANS.ASM
; Block Transfer
; TEXT VRAM Transfer & Restore
;-------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG   100H
;------------------------------------
START:
        MOV AX,0A000H
        MOV DS,AX               ; DS <=0A000H
        MOV AX,0A100H
        MOV ES,AX               ; ES <=0A100H
        MOV SI,0                ; Source ad
        MOV DI,0                ; Dest ad
        MOV CX,4000             ; No.of bytes
        CLD                     ; Clear flag
    REP MOVSB                   ; Move byte
;------------------------------------
DOS:
        MOV AH,0
        INT 21H              ; Return to DOS
;------------------------------------
CODE    ENDS
        END START
```

転送元（MOV DS,AX／MOV SI,0）
転送先（MOV ES,AX／MOV DI,0）
転送バイト数（MOV CX,4000）
CLD ― アドレスを増やす
REP MOVSB ― バイト単位で転送
4000バイトを転送

■ テキスト画面復帰プログラム

テキスト画面を復帰するには、退避プログラムの転送元と転送先のアドレスを入れ替えるだけという簡単なものになります。ついでに、ワード単位（2バイトずつ）の転送も行いましょう。変更箇所は、次のとおりです。

・REP MOVSB を REP MOVSW にする。
・MOV CX. 4000 を MOV CX. 2000 にする。

●ワード単位での転送時には、CX に入れる値に注意。

転送する値を半分にすることを忘れないでください。4000 のままでワード単位の転送をすると 8000 バイトも転送することになり、よけいなものまで動かすことになります。

次に、そのプログラムを挙げます。

■テキスト画面復帰プログラム

```
;--------------------------------
; REST.ASM
; Block Restore
; TEXT VRAM Transfer & Restore
;--------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG    100H
;-------------------------------------
START:
        MOV AX,0A100H
        MOV DS,AX               ; DS <=0A100H
        MOV AX,0A000H
        MOV ES,AX               ; ES <=0A000H
        MOV SI,0                ; Source ad
        MOV DI,0                ; Dest
        MOV CX,2000             ; No.of bytes
        CLD                     ; Clear flag
    REP MOVSW                   ; Move word
;---------------------------------------
DOS:
        MOV AH,0
        INT 21H            ; Return to DOS
;---------------------------------------
CODE    ENDS
        END START
```

転送元・先を入れ替える（MOV AX,0A100H／MOV DS,AX／MOV AX,0A000H／MOV ES,AX）

転送ワード数（MOV CX,2000）

アドレスを増やす（CLD）

ワード単位で転送（REP MOVSW）

2000ワード（4000バイト）をもとに戻す

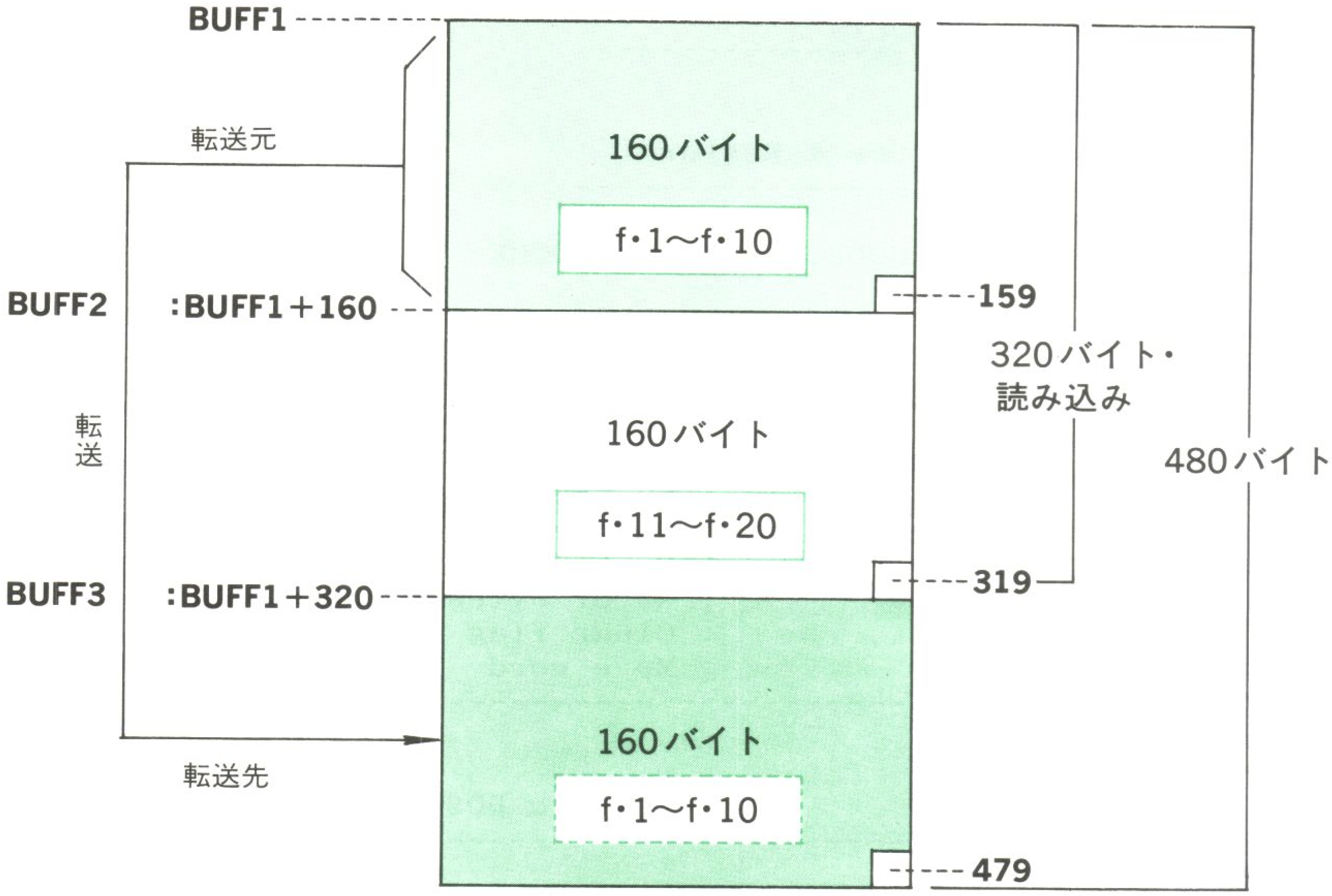
■ファンクションキーの内容入れ替えプログラム

では、ファンクションキーの内容を入れ替えるプログラムを作りましょう。ファンクションキーの読み書きの方法は、前に説明したとおりです。

プログラムのポイントは、次のとおりです。

- **読み込みは、次のようにすべてのキーの内容をBUFF1からBUFF2までに格納するようにする。**

■キー内容読み込みとf･1〜f･10の転送

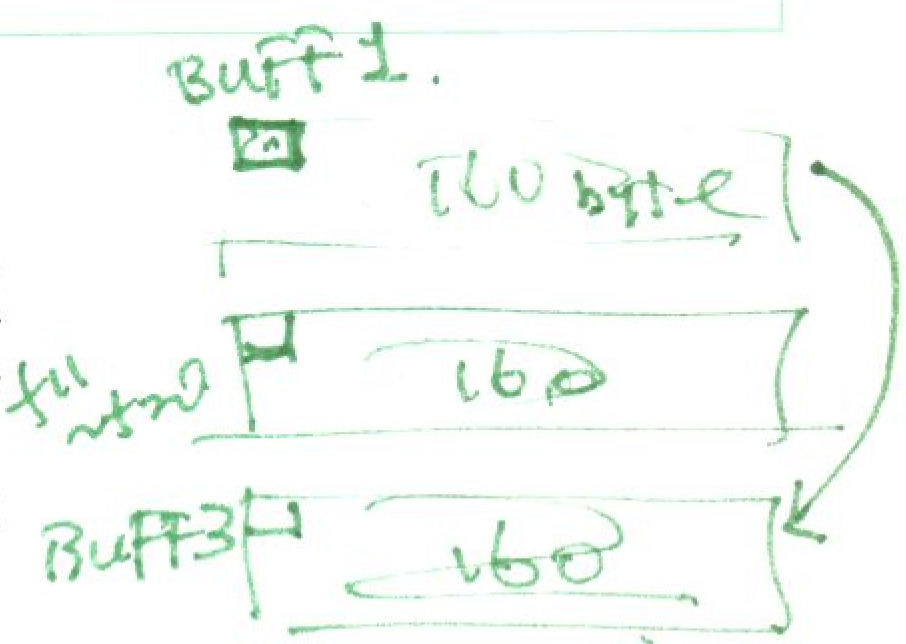


・f・1〜f・10の内容をBUFF3にブロック転送する。
・書き込みは、BUFF2から行うようにする。
・するとキーの内容が入れ替わることになる。

次に、そのプログラムを示します。

■ブロック転送でのファンクションキー入れ替え(1バイト単位)

```
;---------------------------------
; FSWAP3.ASM
; Function Keys Swap 3
; f・1-f・10 <=>f・11-f・20
;---------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG    100H
;----------------------------------------
START:
READ:   MOV DX,OFFSET BUFF1; Fkeys Read      ┐
        MOV CL,0CH         ; buffer          │ ファンクションキー読み込み
        CALL FKEYS         ; Read            ┘
;----------------------------------------
MOVE:
        MOV SI,OFFSET BUFF1; Source ad       ┐ 各先頭アドレス・セット
        MOV DI,OFFSET BUFF3; Dest ad         ┘
        MOV CX,160         ; No.of bytes ──── 転送するバイト数
        CLD                ; Clear flag
    REP MOVSB              ; Move byte ────── バイト単位でのブロック転送
;----------------------------------------
SWAP:
        MOV DX,OFFSET BUFF2; Change offset   ┐
                           ; ad              │ ファンクションキー書き込み
        MOV CL,0DH                           │
        CALL FKEYS         ; Write           ┘
;----------------------------------------
DOS:
        MOV AH,0                             ┐ MS-DOSに戻る
        INT 21H         ; Return to DOS      ┘
;========================================
FKEYS:
        MOV AX,0           ; All Fkeys       ┐ ファンクションキー
        INT 0DCH           ; Fkeys Set       │   読み書きサブルーチン
        RET                ; RETURN          ┘
;========================================
BUFF1   DB 160  DUP (?)    ; 480 BYTES       ── f・1～f・10のバッファ確保
BUFF2   DB 160  DUP (?)    ; RESERVED        ── f・11～f・20のバッファ確保
BUFF3   DB 160  DUP (?)                      ── 転送先のバッファ確保
CODE    ENDS
        END START
```



COM ファイルでは、その実行時に CS, DS, ES のセグメントの値が同じになります。そのため、プログラムのなかでセグメント・レジスタに値を設定していません。もし、そのことをはっきりと指定したければ、次の命令を追加してもかまいません。

```
MOV AX, CS
MOV DS, AX
MOV ES, AX
```

この例では、転送元のセグメントと転送先のセグメントは同じことになります。

■ 2バイト単位でのブロック転送

ブロック転送でMOVSWを使って、2バイト単位で転送するプログラムは、次のようになります。この例では、シンボル（ラベル）を使っています。

■ブロック転送でのファンクションキー入れ替え（2バイト単位）

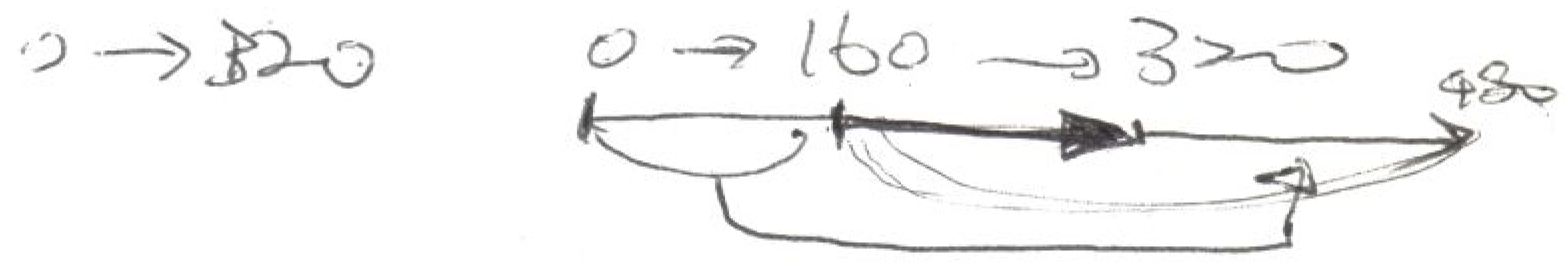
```
;--------------------------------
; FSWAP4.ASM
; Function Keys Swap 4
; f･1-f･10 <=>f･11-f･20
;--------------------------------
FREAD    EQU 0CH                                ┐
FWRITE   EQU 0DH                                │ シンボル定義
ALLKEYS  EQU 0                                  │
FSET     EQU 0DCH                               ┘
NO       EQU 80        ; No. of words:80          80ワード(160バイト)
;--------------------------------------
CODE     SEGMENT
         ASSUME CS:CODE,DS:CODE,ES:CODE
         ORG    100H
;--------------------------------------
START:
READ:    MOV DX,OFFSET BUFF1; Fkeys Read
         MOV CL,FREAD       ; buffer
         CALL FKEYS         ; Read
;--------------------------------------
MOVE:
         MOV SI,OFFSET BUFF1; Source ad
         MOV DI,OFFSET BUFF1+320 ──────── こういうふうに＋で
                            ; Dest ad       アドレスを指定可
         MOV CX,NO          ; No.of words
         CLD                ; Clear flag
     REP MOVSW              ; Move word ──── ワード(2バイト)単位での
;--------------------------------------         ブロック転送
SWAP:
         MOV DX,OFFSET BUFF1+160
                            ; Change offset ad
         MOV CL,FWRITE
         CALL FKEYS         ; Write
```

```
;-----------------------------------------
DOS:
        MOV AH,0
        INT 21H             ; Return to DOS
;=========================================
FKEYS:
        MOV AX,ALLKEYS        ; All Fkeys
        INT FSET              ; Fkeys Set
        RET                   ; RETURN
;=========================================
BUFF1   DB 480  DUP (?)       ; 480 BYTES
CODE    ENDS
        END START
```

ファンクションキー読み書きサブルーチン

バッファを1度に160×3＝480確保している

また、読み書きのバッファを1度に480バイト取って、各バッファの先頭アドレスをBUFF1から160バイトを加えた指定方法をとっていることにも注目してください。これを前のプログラムと比較すると、次のようになります（P186参照）。



BUFF2：BUFF1＋160

BUFF3：BUFF1＋320

■デバッガで転送結果を確認

```
A>SYMDEB FSWAP4.COM ⏎
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-U ⏎
3952:0100 BA2501          MOV     DX,0125
3952:0103 B10C            MOV     CL,0C
3952:0105 E81700          CALL    011F
3952:0108 BE2501          MOV     SI,0125
3952:010B BF6502          MOV     DI,0265
3952:010E B95000          MOV     CX,0050
3952:0111 F3              REPZ
3952:0112 A5              MOVSW
-U ⏎
3952:0113 BAC501          MOV     DX,01C5
3952:0116 B10D            MOV     CL,0D
3952:0118 E80400          CALL    011F
3952:011B B400            MOV     AH,00
3952:011D CD21            INT     21
3952:011F B80000          MOV     AX,0000
3952:0122 CDDC            INT     DC
3952:0124 C3              RET
-


-
-G11B ⏎
AX=0000  BX=0000  CX=000D  DX=01C5  SP=FFFE  BP=0000  SI=01C5  DI=0305
DS=3952  ES=3952  SS=3952  CS=3952  IP=011B   NV UP EI PL NZ NA PO NC
3952:011B B400            MOV     AH,00
```

BUFF1のアドレス

BUFF1＋320のアドレス
320は16進で140

MS-DOSに戻る1歩手前まで実行

```
-
-D125,2FF ⏎
3952:0120                 FE 20 43-31 20 20 1B 53 00 00 00        ~ C1  .S...
3952:0130  00 00 00 00 00 FE 20 43-55 20 20 1B 54 00 00 00   .....~ CU  .T...
3952:0140  00 00 00 00 00 FE 20 43-41 20 20 1B 55 00 00 00   .....~ CA  .U...
3952:0150  00 00 00 00 00 FE 20 53-31 20 20 1B 56 00 00 00   .....~ S1  .V...
3952:0160  00 00 00 00 00 FE 20 53-55 20 20 1B 57 00 00 00   .....~ SU  .W...
3952:0170  00 00 00 00 00 FE 56 4F-49 44 20 1B 45 00 00 00   .....~VOID .E...
3952:0180  00 00 00 00 00 FE 4E 57-4C 20 20 1B 4A 00 00 00   .....~NWL  .J...
3952:0190  00 00 00 00 00 FE 49 4E-53 20 20 1B 50 00 00 00   .....~INS  .P...
3952:01A0  00 00 00 00 00 FE 52 45-50 20 20 1B 51 00 00 00   .....~REP  .Q...
3952:01B0  00 00 00 00 00 FE 20 5E-5A 20 20 1B 5A 00 00 00   .....~ ^Z  .Z...
3952:01C0  00 00 00 00 00 64 69 72-20 61 3A 0D 00 00 00 00   .....dir a:.....
3952:01D0  00 00 00 00 00 64 69 72-20 62 3A 0D 00 00 00 00   .....dir b:.....
3952:01E0  00 00 00 00 00 63 6F 70-79 20 00 00 00 00 00 00   .....copy ......
3952:01F0  00 00 00 00 00 64 65 6C-20 00 00 00 00 00 00 00   .....del .......
3952:0200  00 00 00 00 00 72 65 6E-20 00 00 00 00 00 00 00   .....ren .......
3952:0210  00 00 00 00 00 63 68 6B-64 73 6B 20 61 3A 0D 00   .....chkdsk a:..
3952:0220  00 00 00 00 00 63 68 6B-64 73 6B 20 62 3A 0D 00   .....chkdsk b:..
3952:0230  00 00 00 00 00 74 79 70-65 20 00 00 00 00 00 00   .....type ......
3952:0240  00 00 00 00 00 64 61 74-65 0D 00 00 00 00 00 00   .....date.......
3952:0250  00 00 00 00 00 74 69 6D-65 0D 00 00 00 00 00 00   .....time.......
3952:0260  00 00 00 00 00 FE 20 43-31 20 20 1B 53 00 00 00   .....~ C1  .S...
3952:0270  00 00 00 00 00 FE 20 43-55 20 20 1B 54 00 00 00   .....~ CU  .T...
3952:0280  00 00 00 00 00 FE 20 43-41 20 20 1B 55 00 00 00   .....~ CA  .U...
3952:0290  00 00 00 00 00 FE 20 53-31 20 20 1B 56 00 00 00   .....~ S1  .V...
3952:02A0  00 00 00 00 00 FE 20 53-55 20 20 1B 57 00 00 00   .....~ SU  .W...
3952:02B0  00 00 00 00 00 FE 56 4F-49 44 20 1B 45 00 00 00   .....~VOID .E...
3952:02C0  00 00 00 00 00 FE 4E 57-4C 20 20 1B 4A 00 00 00   .....~NWL  .J...
3952:02D0  00 00 00 00 00 FE 49 4E-53 20 20 1B 50 00 00 00   .....~INS  .P...
3952:02E0  00 00 00 00 00 FE 52 45-50 20 20 1B 51 00 00 00   .....~REP  .Q...
3952:02F0  00 00 00 00 00 FE 20 5E-5A 20 20 1B 5A 00 00 00   .....~ ^Z  .Z...
-Q ⏎

A>
```

125H(BUFF1)

265H(BUFF1＋320)
320は16 進で140

転送されている

第3部

第3章 独自の実用コマンドを作る

マシン語でプログラムを開発すると、いろいろと実用的なものになるものです。ここでは、MS-DOSのファイル読み書きのファンクション・コールを学びながらちょっとした実用プログラムを作ってみましょう。

3・1 アスキーファイル縮小印刷プログラム

■ 縮小印刷プログラムの内容

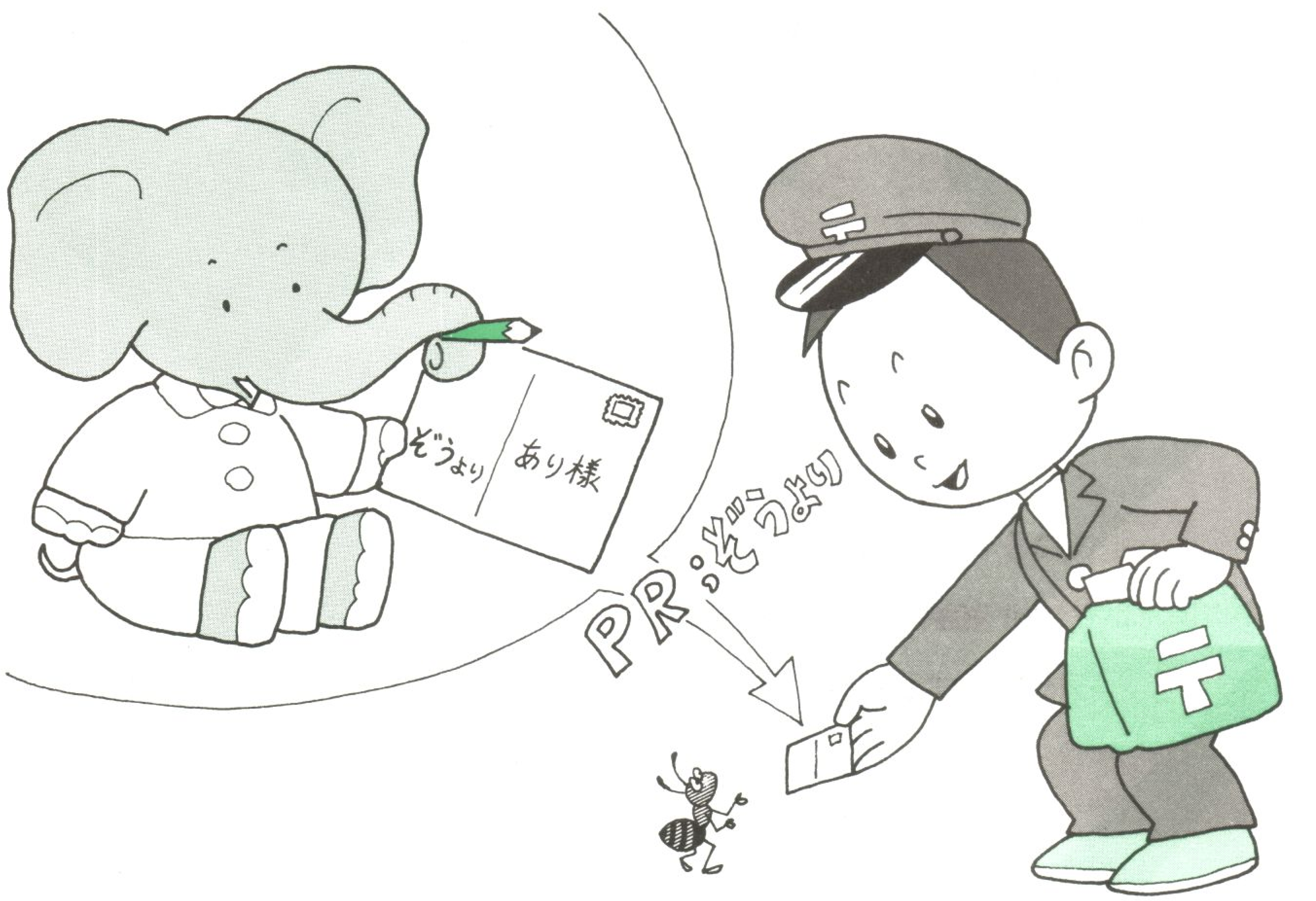
これは、英数字だけのアスキーファイルを読み込んで、その内容を縮小文字で印刷するものです。本書でのアセンブリ言語のソースファイルなど、小さく印刷したほうが便利なときに利用できます。横が 80 桁しか打てないプリンタでも 136 桁打てますし、縦も 1 枚では入り切れないものも、ちゃんと入るようになります。

1 例として、ここで紹介する縮小印刷プログラムのソース・リストを縮小印刷してみたのが、次のものです(原寸大)。COM ファイルは、PR. COM としますので実行のしかたは、

● PR は PRINT の省略で付けたファイル名。

PR ファイル名

とします。



■縮小印刷例(原寸大)

```
;-------------------------------------
; PR.ASM
; Print ASCII File in Condensed Mode
; Usage: PR filename
;------------------------------------
CODE   SEGMENT
       ASSUME CS:CODE,DS:CODE,ES:CODE
       ORG 80H         ; Work area
CMDLN  DB 128 DUP (?) ;Command line buffer
;
       ORG   100H
;---------------------------------------
START:
;---------------------------------------
CHECK:
       MOV BL,CMDLN[0] ; If no filename
       CMP BL,0        ; then ERROR
       JE ERROR
       MOV BH,0           ; ASCIZ for
       MOV CMDLN[BX+1],0  ; File Open
;---------------------------------------
CONDENSED:               ; Condensed Mode
       MOV SI,OFFSET CND_SET
       CALL LPRINT
;---------------------------------------
FOPEN:                   ; File Open
       MOV AH,3DH
       MOV DX,OFFSET CMDLN[2]
       MOV AL,0        ; Read mode
       INT 21H
       JC ERROR        ; Error then ERROR
       MOV BX,AX    ; BX <= File handle
;=======================================
CND_SET   DB 27,'T18'   ; Condensed
          DB 27,'Q',13,10,0
RESET     DB 27,'A'     ; Normal
          DB 27,'H',13,10,0
BUFF      DB 1 (?)
ERROR_MSG DB "File can't open.$"

CODE   ENDS
       END START
```

最近はやりのシステム手帳のリフィルにぴったりのサイズです。こうしていつでもアセンブリ言語のリストを持ち歩くと上達が速くなることうけあいです。なお、このプログラムは、なるべく簡単でわかりやすくするため、漢字が入ったファイルは対象外としています。

■ここでのポイント

縮小印刷プログラムを作るには、次の点についてプログラミングの方法を知っておかなければなりません。

- **ファイルのオープン**
- **ファイルの読み出し**
- **ファイルのクローズ**
- **プリンタ出力**

まずは、ファイル関係から始めましょう。

■ファイルのオープン

BASIC 言語では、シーケンシャルやランダムファイルをオープンするときは、次のようにファイルバッファの番号を使って行うことは、ご承知のとおりです。

・**読み出し**

```
OPEN "TEST. DAT"  FOR INPUT AS #1
```

・**書き込み**

```
OPEN "TEST2. DAT"  FOR OUTPUT AS #2
```

・**読み書き**

```
OPEN "TEST3. DAT"  AS #3
```

●MS-DOS でのファイル読み書きは、特別なファイル番号であるファイルハンドルを使って行う。

MS-DOS でも、ファイルを読み書きするときにファイル番号を使うことができます。この番号をファイルハンドル(File Handle：ファイルを取り扱うもの）といいます。ここでは、ファイルハンドルを使ったファイル・オープンのファンクション・コールを紹介します。ここでは、いろいろと決まりごとを理解しておかなければいけません。すこし、面倒かも知れませんが、知っておかなければどうしようもないことです。

＜ファイル・オープン＞

●ファイル・オープンのファンクション・コール。

実行：

AH ← 3DH

DX ←ファイル名の先頭アドレス（ファイル名の最後は 0 で終わっておかなければならない）

AL ← オープンの状態

0：読み出しのためにオープンする

1：書き込みのためにオープンする

2：読み書きのためにオープンする

INT 21 H

結果：

CF＝0 のとき正常終了で、

AX にファイルハンドルの値が入る。

CF＝1 のときが異常終了。

AX にエラーの状態に応じて 2 から 5, 0CH の値が入る。ここでは、AX が 0 以外は、オープンエラーと考えてください。

● CF は、Carry Flag(キャリーフラグ)で桁上がり、桁下がりのときに立つフラグ。ここでは、異常を示すフラグとして使ってある。

ここで、BASIC 言語とは違う点を挙げます。

- オープンのしかたは、シーケンシャルファイルでもランダムファイルでも同じ。
- ファイルハンドルは、ファイルをオープンすると MS-DOS が AX に入れてユーザーに与える。ユーザーが勝手には決められない。この与えられた値は、知っておく必要はない。
- オープンが正常に行われたかどうかは、ユーザーが判定しなくてはならない（CF や AX の内容を見て）。
- ファイルハンドルの値は、ファイルをクローズするまで決められたレジスタ（BX）に入れておかなければならない。

▶ファイル・オープンの例

ファイル名 TEST. DAT を読み出しのためにオープンする例を挙げます。

＜1. ファイル名が決まっている場合＞

●JC は、Jump on Carry でキャリーフラグが立ったら、ジャンプする。

```
        MOV AH, 3DH
        MOV DX, OFFSET FNAME
        MOV AL, 0
        INT 21H
        JC ERROR
        MOV BX, AX

FNAME DB 'TEST. DAT', 0
```

ポイントは、次のとおりです。

- ファイル名が決まっているとき、プログラムの中に入れる場合。
- ファイル名の最後に 0 を付けておく。
- エラーとなったら CF が立つ。その判断と処理は、JC（Jump on Carry）でエラー処理ルーチン ERROR にジャンプしている。
- 正常にオープンしたら、BX に AX のファイルハンドルの値を入れる。

＜2. ファイル名をプログラムの実行時に入力する場合＞

TYPE コマンドのようにコマンドファイル実行時にファイル名を入力するには、ちょっと複雑になります。例えば、ファイルの内容を表示するプログラム名を FTYPE.COM とし、表示するファイル名を TEST. DAT とすると、

```
A>FTPYE TEST. DAT
```

●これは、MS-DOS が内部で処理を行うもので、こういうふうにファイル名を格納します。

とコマンドラインで、表示するファイル名を入力します。

このとき、このファイル名は、コマンドファイルのコードセグメントのオフセット 80H から、次のように格納されます。

■ファイル名の格納のされ方



・最初の数は、ファイル名の文字数を表わす。
・ファイル名の最後は、CR が入っている。

そこで、オフセット 80H に CMDLN というシンボルをつけて、格納されているファイル名をファイル・オープンのために処理をします。

● CMDLN は、CoMmanD LiNe の略。

```
          ORG 80H

CMDLN DB 128 DUP  (?)

          ORG 100H

          MOV BL, CMDLN [0]
          CMP BL, 0
          JE ERROR
          MOV BH, 0
          MOV CMDLN [BX+1], 0

          MOV DX, OFFSET CMDLN [2]
          MOV AL, 0
          INT 21 H
          JC ERROR
          MOV BX, AX
```

以下、この説明をします。

```
ORG 80H
```

● DUP(?)の(?)は、メモリの内容はそのままで、とにかくメモリを確保する意味。

```
CMDLN DB 128 DUP (?)
```

●○○○○○ ——
128バイト

で、80HからCMDLINという名前で128バイト分を確保しています。これはコマンドラインのバッファが128バイトあるからです。

```
ORG 100H
```

● CMDLIN[0]は、CMDLINの指すアドレスに0を足したアドレスの内容を示す(ここではアドレス80Hの内容)。

```
MOV BL, CMDLN [0]
CMP BL, 0
JE ERROR
```

で、BLにはCMDLNのはじめの値（ファイル名の文字数）が入ります。それが、0であれば、ファイル名の入力がなかったということでエラー処理にジャンプさせます。

● CMDLN［BX＋1］は、CMDLNの指すアドレスにBXの指すアドレス+1したアドレスの内容を示す(ここでは、80H+9+1で8AH)。

```
MOV BH, 0
MOV CMDLN [BX+1] , 0
```

で、BXにはファイル名の文字数が入り、それに+1したアドレスがファイル名の終わりですから、そこに0を書き込みます。このようにファイル名のとその終わりを示すのに、最後に0をつけたものをASCIZ(ASCII＋Zero：アスキーゼロ)文字列といいます。

●アスキーコードで表現された文字列の最後に1バイトの0を付加したものをASCIZ文字列という。

● OFFSET CMDLIN［2］は、CMDLINの指すアドレスにCMDLIN+2したアドレスを示す(ここではアドレス82H)。

```
MOV AH, 3DH
MOV DX, OFFSET CMDLN [2]
MOV AL, 0
INT 21H
JC ERROR
MOV BX, AX
```

で、DXにファイル名の先頭アドレス（CMDLN＋2）がセットされ、ファイルがオープンできるようになります。

ファイルの読み出し

ファイル（シーケンシャルファイル）の読み出し方は、次のとおりです。

●ファイル読み出しのファンクション・コール。

実行：

　AH ← 3FH
　DX ←読み出しバッファの先頭アドレス
　CX ←読み出すバイト数
　BX ←ファイルハンドルの値
　INT 21 H

結果：

　CF=0で正常終了。AXに読み出されたバイト数が入る。
　CF=1のときはエラー。

1バイト読み出すプログラムは、次のようになります。

```
        MOV AH, 3FH
        MOV DX, OFFSET BUFF
        MOV CX, 1
        INT 21H
        JC ERROR

BUFF DB 1 (?)
```

このように、読み出しは、ユーザーがバッファを決め、バイト数を決めて行います。なお、BXにはファイルハンドルが入っていることを常に確認しておく必要があります。

ファイルのクローズ

ファイルのクローズのしかたは、次のとおりです。

●ファイル・クローズのファンクション・コール。

実行：

AH ← 3EH

BX ←ファイルハンドルの値

INT ← 21H

結果：

CF＝0で正常クローズされた。CF＝1のときはエラー。

このプログラム例は、次のようになります。

```
MOV AH, 3EH
INT 21H
```

BX にはファイルハンドルが入っていることを確認。

■ファイル表示プログラム

これまでの知識をもとに、練習としてファイル表示プログラムを作ってみましょう。これは、TYPE コマンドと同じ機能をもつものです。次に、そのリストを挙げます。



■ファイル表示プログラム

```
;-------------------------------------
; FTYPE.ASM
; Type ASCII File
; Usage: FTPYE filename
;-------------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG 80H          ; Work area
CMDLN   DB 128 DUP (?) ;Command line buffer    ── ファイル名が
                                                  格納されるエリア
;
        ORG   100H
;----------------------------------------
START:
;----------------------------------------
CHECK:
        MOV BL,CMDLN[0] ; If no filename       ┐ ファイル名が入力
        CMP BL,0        ; then ERROR           ┘ されなかったらエラー処理へジャンプ
        JE ERROR
        MOV BH,0           ; ASCIZ for         ┐
        MOV CMDLN[BX+1],0  ; File Open         ┘ ファイル名の最後に0を入れる
;----------------------------------------
FOPEN:                   ; File Open
        MOV AH,3DH                             ┐
        MOV DX,OFFSET CMDLN[2]                 │ ファイル・オープン
        MOV AL,0         ; Read mode           │ (読み込みモード)
        INT 21H                                ┘
        JC ERROR         ; Error then ERROR    ── エラー処理へ
        MOV BX,AX     ; BX <= File handle      ── BXにファイルハンドルを入れる
;----------------------------------------
READ:                    ; File read
        MOV AH,3FH                             ┐
        MOV DX,OFFSET BUFF                     │ 1バイトを読んで
        MOV CX,1         ; One byte read       │ BUFFの中に入れる
        INT 21H          ; & Store at BUFF     ┘
        CMP AX,0         ; If no read          ┐
        JE FCLOSE        ; then File Close     ┘ データがなければファイル・クローズ
        MOV DL,BUFF      ; If End of File      ┐ 最後のデータが
        CMP DL,1AH       ; then File Close     │ 1AHであれば
        JE FCLOSE                              ┘ ファイル・クローズ
;----------------------------------------
 PRINT:
        MOV DL,BUFF                            ┐
        MOV AH,2                               │ 1文字表示
        INT 21H                                │
        JMP SHORT READ   ; Go back to read     ┘----- 1バイト読み込みへ戻る
;----------------------------------------
FCLOSE:                  ; File Close
        MOV AH,3EH                             ┐ ファイル・クローズ
        INT 21H                                ┘
;----------------------------------------
DOS:
        MOV AH,0                               ┐ MS-DOSへ戻る
        INT 21H          ; Return to DOS       ┘
;----------------------------------------
ERROR:
        MOV DX,OFFSET ERROR_MSG                ┐
        MOV AH,9         ; Print Error         │----- エラーメッセージ表示
        INT 21H          ; Message             ┘
        JMP SHORT DOS                          ── DOSへジャンプ
BUFF        DB 1 (?)                           ── ファイルから読んだ1バイトを格納するところ
ERROR_MSG DB 'File can't open.$'  ── エラーメッセージ

CODE    ENDS
        END START
```

Short ±128バイト以内でジャンプする。

プログラムのポイントは、次のとおりです。

- ファイル名がなければ、エラーメッセージを表示して終了。
- 読み込みバッファ（BUFF）を1バイト確保して、1バイトずつファイルから読み出している。
- その1バイトを1文字表示で画面表示している。

```
MOV DL, BUFF
MOV AH, 2
INT 21
```

- MOV DL, BUFF で直接、メモリからレジスタに内容を入れることができる。
- AX の値が0であれば、読み込みを終了する。
- 読み込んだ1バイトがファイルの終わりを示す1AH ならば、読み込みを終了する。テキストファイルの終わりは、必ず1AH となっている。

● 1AH は、テキストファイルの終わりを示すアスキーコード。



次に、実行例を示します。

```
A>FTYPE DISP.BAT ⏎
File can't open.  ――ファイル名が間違っていたのでエラー表示
A>FTYPE FDISP.BAT ⏎
ECHO OFF
CLS
ECHO ********** ファイル表示 **********
IF "%1"=="" GOTO END
:START
  ECHO ドライブBに表示したいディスクを入れてください.
  PAUSE
  ECHO ----------------------------------------------
  DIR B: ¦ FIND "%1"
REM --- ＹＥＳ／ＮＯ判断 ---
   ECHO 処理を続けますか（Ｙ／Ｎ）？
   YESNO3
   IF ERRORLEVEL 1 GOTO END
   ECHO ---------------------------------------------
   GOTO START
:END

A>
```

ファイルの内容が表示される

■プリンタ出力

ファイルをプリンタに出力するには、基本的には、ファイル表示プログラムの１文字表示の部分を、プリンタに１文字出力するように変更すればOKです。プリンタ１文字出力のファンクション・コールは、次のとおりです。

```
MOV DL, 出力する文字
MOV AH, 5
INT 21 H
```

●プリンタ１文字出力のファンクション・コール。

縮小印刷するには、縮小文字にする制御コードをプリンタに送る必要があります。しかし、コードを１つずつ送っていたのでは効率が悪いので、次のように複数の文字を出力するサブルーチンを作りましょう。

●プリンタ複数文字出力のサブルーチン。

```
LPRINT：
        MOV DL, [SI]
        CMP DL, 0
        JE LPRINT_END
        MOV AH, 5
        INT 21 H
        INC SI
        JMP SHORT LPRINT
LPRINT_END：
        RET
```

呼び出し方は、次のとおりです。

- SI に出力する文字列の先頭アドレスを入れる。
- その文字列の終わりは、0 とする（ASCIZ）。
- CALL LPRINT で呼ぶ。

サブルーチンのポイントは、次のとおりです。

- 文字列の終わり 0 を CMP で判定している。
- 文字列のアドレスを SI で指定し、1 文字を出力するたびに＋1している。
- JMP SHORT というのは、＋－128 バイト以内の範囲でジャンプする命令（SHORT は疑似命令）で、アセンブルしたときは、この命令に対するマシン語のバイト数が JMP だけのときより少なくなる。また、実行速度もわずかながら速い（人間にはわからないが）。

●SHORT を付けない JUMP では、飛び先の範囲に制限はない。

制御コードは、次のようにして格納します（NEC 系のプリンタの場合）。

```
CND_SET DB 27, 'T18'
        DB 27, 'Q', 13, 10, 0
```

これは BASIC 言語で、

LPRINT CHR$(27) ; "T18"
LPRINT CHR$(27) ; "Q"

とするのと同じで、行間を狭くし、縮小モードにしています。

■縮小印刷プログラム

ようやく、目的とするプログラムの開発までに至りました。これまでのことを総合すれば、作ることができます。そのプログラムは、次のとおりです。また、内容も理解できることと思います。

■縮小印刷プログラム

```
;-------------------------------------
; PR.ASM
; Print ASCII File in Condensed Mode
; Usage: PR filename
;-------------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG 80H           ; Work area
CMDLN   DB 128 DUP (?) ;Command line buffer  ── ファイル名が格納されるエリア
;
        ORG   100H
;---------------------------------------
START:
;---------------------------------------
CHECK:
        MOV BL,CMDLN[0]  ; If no filename     ┐ ファイル名がなかったら
        CMP BL,0         ; then ERROR         │ エラー処理へジャンプ
        JE ERROR                              ┘
        MOV BH,0            ; ASCIZ for       ┐ ファイル名の最後に
        MOV CMDLN[BX+1],0   ; File Open       ┘ 0を入れる
;---------------------------------------
CONDENSED:                ; Condensed Mode    ┐ 縮小コードを
        MOV SI,OFFSET CND_SET                 │ プリンタに送る
        CALL LPRINT                           ┘
;---------------------------------------
FOPEN:                    ; File Open         ┐
        MOV AH,3DH                            │ ファイル・オープン
        MOV DX,OFFSET CMDLN[2]                │ (読み込みモード)
        MOV AL,0         ; Read mode          ┘
        INT 21H
        JC ERROR         ; Error then ERROR  ── エラー処理へ
        MOV BX,AX     ; BX <= File handle    ── BXにファイルハンドルを入れる
;---------------------------------------
```

```
READ:                         ; File read              1バイト読んでBUFFの
        MOV AH,3FH                                     中に入れる
        MOV DX,OFFSET BUFF
        MOV CX,1              ; One byte read
        INT 21H               ; & Store at BUFF
        CMP AX,0              ; If no read             データがなければ
        JE FCLOSE             ; then File Close        終わり
        MOV DL,BUFF           ; If End of File         最後のデータが
        CMP DL,1AH            ; then File Close        1AHであれば終わり
        JE FCLOSE
;---------------------------------------
PRINTER:                      ; Send one byte
        MOV DL,BUFF           ; to Printer             1文字プリンタ出力
        MOV AH,5
        INT 21H
        JMP SHORT READ        ; Go back to read ——— 1バイト読み込みに戻る
;---------------------------------------
FCLOSE:                       ; File Close             ファイル・クローズ
        MOV AH,3EH
        INT 21H
;---------------------------------------
PRESET:                       ; Printer Reset          プリンタを初期設定に戻す
        MOV SI,OFFSET RESET                            (通常印字モード)
        CALL LPRINT
;---------------------------------------
DOS:
        MOV AH,0                                       MS-DOSに戻る
        INT 21H               ; Return to DOS
;---------------------------------------
ERROR:
        MOV DX,OFFSET ERROR_MSG
        MOV AH,9              ; Print Error            エラーメッセージ表示
        INT 21H               ; Message
        JMP SHORT PRESET ——— PRESETへジャンプ
;=======================================
LPRINT:                       ; Lprint SUB ——— 制御コード出力サブルーチン
        MOV DL,[SI] ——— データをDLに入れる
        CMP DL,0 ——— データがなければ終了
        JE LPRINT_END
        MOV AH,5              ; Send one byte          1バイトをプリンタ
        INT 21H               ; to Printer             に出力
        INC SI ——— データのアドレスを1つ増やす
        JMP SHORT LPRINT ——— はじめに戻る
LPRINT_END:
        RET                   ; Return ——— メインルーチンに戻る
;=======================================
CND_SET    DB 27,'T18'        ; Condensed ——— 行間設定
           DB 27,'Q',13,10,0 ——— 縮小モード
RESET      DB 27,'A'          ; Normal                 初期設定
           DB 27,'H',13,10,0
BUFF       DB 1 (?) ——— ファイルから読み込んだ1バイトを格納するところ
ERROR_MSG DB "File can't open.$" ——— エラーメッセージ

CODE    ENDS
        END START
```

3·2 画面表示保存プログラム

■ 画面表示保存プログラムの内容

画面に表示された結果をファイルとして保存できると、いろいろなメリットが生じます。例えば、次のようなことです。

・画面情報の再利用が可能となる。
・内容の加工が可能となる。
（ハード・コピーでは、これらができない。）
・必要とあれば、すぐファイル内容が印字できる。
・画面表示とファイル作成が同時にできる。
（リダイレクションを使って、画面表示をファイル出力に切り替えると、ファイルはできるが画面表示ができなくなる。）

そこで、次のような画面表示保存プログラムを作ることにしましょう。

・保存できる文字は、英数字だけに限定する。
・漢字は対象外とする（複雑になるため）。

ファイル名は、SAVE. COM とします。使い方は、

SAVE ファイル名

とします。
次に、実行例を挙げます。

Q → 51 → 81
T → 54 → 84

＜アセンブラでエラーが出た箇所を保存する例＞

■アセンブル・エラーの箇所を保存

```
A>MASM SAVE ⏎
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

Object filename [SAVE.OBJ]:
Source listing  [NUL.LST]:
Cross reference [NUL.CRF]:
 0122  A3 0000 U                        MOV ED,AX        ; ES <= 0A000H
 E r r o r  ---        9:Symbol not defined
 0150  BA 0000 U                        MOV DX,OFFSET ERROR_MGS
 E r r o r  ---        9:Symbol not defined

49636 Bytes free

Warning Severe
Errors  Errors
0       2

A>SAVE ERROR.DAT ⏎
```

アセンブルでエラーが表示された

この画面をERROR.DATというファイル名で保存する

■保存した内容の表示

```
A>TYPE ERROR.DAT ⏎
A>MASM SAVE
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

Object filename [SAVE.OBJ]:
Source listing  [NUL.LST]:
Cross reference [NUL.CRF]:
 0122  A3 0000 U                        MOV ED,AX        ; ES <= 0A000H
 E r r o r  ---        9:Symbol not defined
 0150  BA 0000 U                        MOV DX,OFFSET ERROR_MGS
 E r r o r  ---        9:Symbol not defined

49636 Bytes free

Warning Severe
Errors  Errors
0       2

A>SAVE ERROR.DAT

A>
```

保存コマンド実行も保存されている

画面が保存されている

■画面保存データを縮小印刷した例

●前出の縮小印刷プログラムで印字したもの。

```
A>MASM SAVE
Microsoft MACRO Assembler  Version 3.00
(C)Copyright Microsoft Corp 1981, 1983, 1984

Object filename [SAVE.OBJ]:
Source listing  [NUL.LST]:
Cross reference [NUL.CRF]:
 0122  A3 0000 U                        MOV ED,AX       ; ES <= 0A000H
 E r r o r  ---       9:Symbol not defined
 0150  BA 0000 U                        MOV DX,OFFSET ERROR_MGS
 E r r o r  ---       9:Symbol not defined

49636 Bytes free

Warning Severe
Errors  Errors
0       2

A>SAVE ERROR.DAT
```

＜バッチ・ファイルのなかで、特定の拡張子をもつファイル名をソートして保存する例＞

■バッチ・ファイルの中での実行例

```
ECHO OFF
CLS
DIR *.ASM ¦ FIND "-" ¦ SORT
SAVE ASM.DAT ──── ソートしたASMファイルを表示・保存する
CLS
DIR *.BAT ¦ FIND "-" ¦ SORT
SAVE BAT.DAT ──── ソートしたBATファイルを表示・保存する
```

```
B>DIRSAVE ⏎

B>ECHO OFF
FASM       ASM       1581   87-07-24   15:30
FASM2      ASM       1622   87-07-24   15:30
FCOM       ASM       1404   87-07-24   14:07
FINIT      ASM       1463   87-07-24   14:30
FKEY       ASM        403   87-07-24   14:42
FSWAP      ASM       1313   87-07-28   17:14
FSWAP2     ASM       1492   87-07-28   17:36
FSWAP3     ASM       1229   87-07-28   18:18
FSWAP4     ASM       1351   87-07-28   19:17
ASM        BAT         64   87-05-17   20:08
DIRSAVE    BAT        107   87-07-29    1:30
DIRSORT    BAT        547   87-07-14    0:00
FDISP      BAT        413   87-07-23    1:01
FILES      BAT        657   87-07-09    1:59
FTYPE      BAT        427   87-07-13   23:26
MNU        BAT       1130   87-07-23    2:18
```

ソートされたASMファイルが表示・保存される

ソートされたBATファイルが表示・保存される

```
B>TYPE ASM.DAT ⏎
FASM       ASM       1581   87-07-24   15:30
FASM2      ASM       1622   87-07-24   15:30
FCOM       ASM       1404   87-07-24   14:07
FINIT      ASM       1463   87-07-24   14:30
FKEY       ASM        403   87-07-24   14:42
FSWAP      ASM       1313   87-07-28   17:14
FSWAP2     ASM       1492   87-07-28   17:36
FSWAP3     ASM       1229   87-07-28   18:18
FSWAP4     ASM       1351   87-07-28   19:17
```

ASMファイルの画面データを表示

```
B>TYPE BAT.DAT ⏎
ASM        BAT         64   87-05-17   20:08
DIRSAVE    BAT        107   87-07-29    1:30
DIRSORT    BAT        547   87-07-14    0:00
FDISP      BAT        413   87-07-23    1:01
FILES      BAT        657   87-07-09    1:59
FTYPE      BAT        427   87-07-13   23:26
MNU        BAT       1130   87-07-23    2:18
```

BATファイル画面データを表示

■ここでのポイント

画面表示保存プログラムを作るためには、次の点についてのプログラミング知識が必要です。

・ファイルの作成
・ファイルへの書き込み
・ファイルのクローズ
・画面情報の取り出し方

以下、各ポイントについて説明します。

■ファイルの作成

BASIC 言語では、新規ファイルをオープンすると、自動的にファイルが作れますが、MS-DOS では、新規ファイルを作るためのファンクション・コールが用意されています。

ファイルハンドルは-ファイルをクローズするまで、BXに！

＜ファイル作成＞

●ファイル作成のファンクション・コール。

実行：

　　AH ← 3CH
　　DX ←ファイル名の先頭アドレス（ファイル名の最後は
　　　　0 で終わっておかなければならない）
　　CX ←ファイルの属性（通常のファイルは 0）
　　INT 21H

結果：

　　CF=0 のとき正常終了で、
　　AX のファイルハンドルの値が入る。
　　CF=1 のとき異常終了で、
　　AX にエラーの状態に応じて 3 から 5 の値が入る。ここでは、AX が 0 以外は、作成エラーと考えてください。

●ファイルの属性とは、読み出しだけのファイルや読み書きができるファイルなどファイルの性質を示すもの。

ここで、ファイル作成のポイントを挙げます。

・新規作成と書き込みのためのファイルオープンが同時に行われる。

・ファイルハンドルは、ファイルを作成すると MS-DOS が AX に入れてユーザーに与える。ユーザーが勝手には決められない。この与えられた値は、知っておく必要はない。
・ファイル作成が正常に行われたかどうかは、ユーザーが判定しなくてはならない（CF や AX の内容を見て）。
・ファイルハンドルの値は、ファイルをクローズするまで決められたレジスタ（BX）に入れておかなければならない。

＜ファイル作成の例＞
ファイル名 TEST. DAT を作成する例を挙げます。

＜1. ファイル名が決まっている場合＞

```
        MOV AH, 3CH
        MOV DX, OFFSET FNAME
        MOV CX, 0
        INT 21H
        JC ERROR
        MOV BX, AX

FNAME DB 'TEST. DAT', 0
```

ポイントは、次のとおりです。

・ファイル名が決まっているとき、プログラムの中に入れる場合は、ユーザーメモリ内に格納する。
・ファイル名の最後に 0 を付けておく。
・エラーとなったら CF が立ち、その判断と処理は、JC（Jump on Carry）でエラー処理ルーチン ERROR にジャンプしている。
・正常にオープンしたら、BX に AX のファイルハンドルの値を入れる。

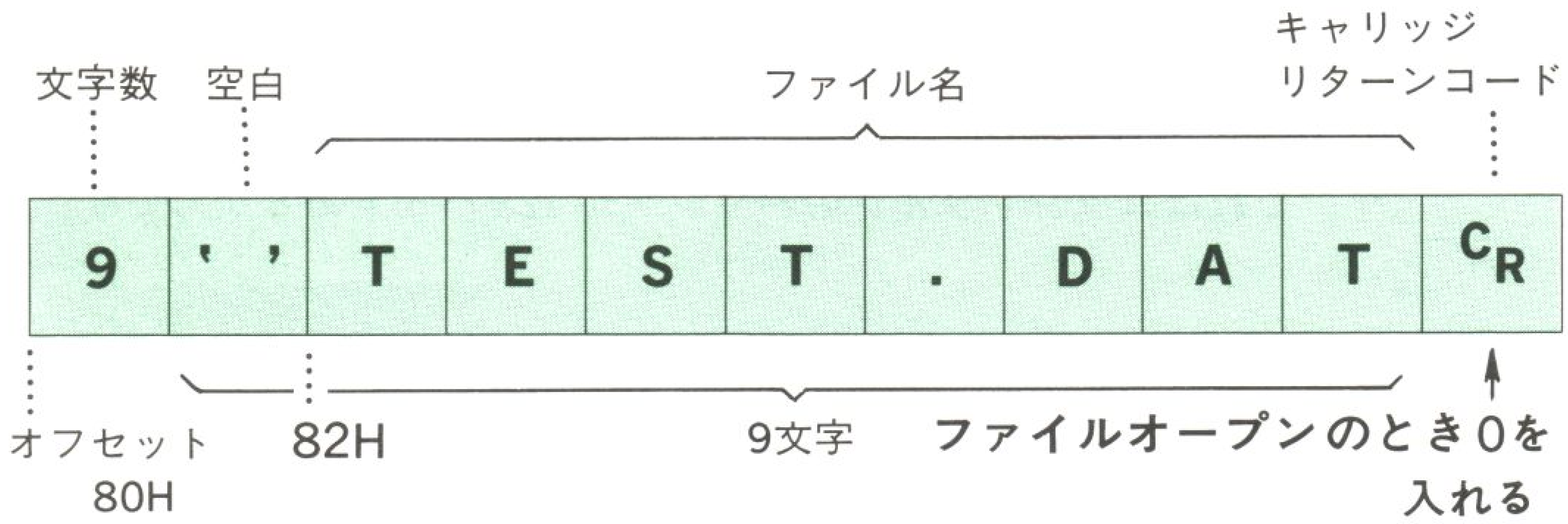
＜2. ファイル名をプログラムの実行時に入力する場合＞
TYPE コマンドのようにコマンドファイル実行時にファイル名を入力するには、ちょっと複雑になります。例えば、保存するファイル名を TEST. DAT とすると、

```
A>SAVE TEST. DAT
```

とコマンドラインで、保存するファイル名を入力します。

このとき、このファイル名は、コマンドファイルのコードセグメントのオフセット 80H から、次のように格納されます。

■ファイル名の格納のされ方



・最初の数は、ファイル名の文字数を表わす。
・ファイル名の最後は、CR が入っている。

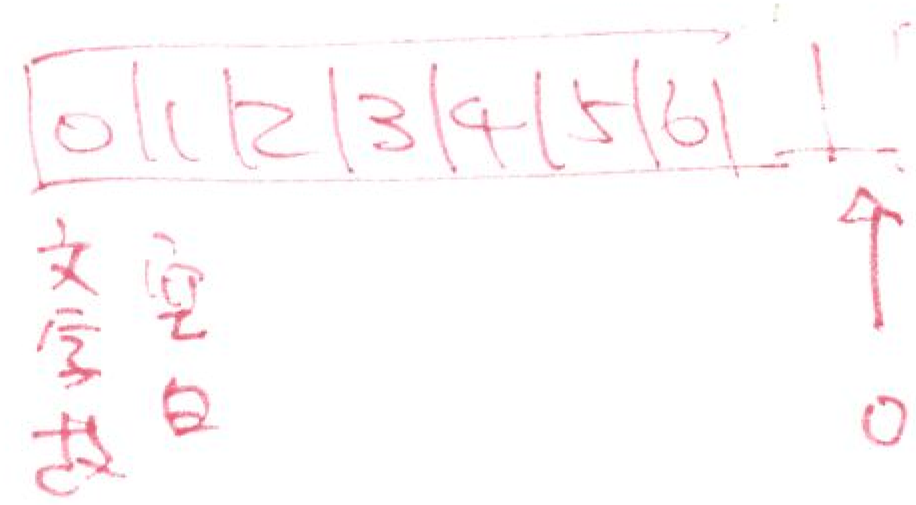
そこで、オフセット 80H に CMDLN というシンボルをつけて、格納されているファイル名でファイルを作成するための処理をします。

```
        ORG 80H
CMDLN   DB 128 DUP (?)

        ORG 100H

        MOV BL, CMDLN [0]
        CMP BL, 0
        JE ERROR
        MOV BH, 0
        MOV CMDLN [BX+1], 0

        MOV DX, OFFSET CMDLN [2]
        MOV CX, 0
        INT 21H
        JC ERROR
        MOV BX, AX
```



以下、この説明をします。

```
        ORG 80H
CMDLN DB 128 DUP （?）
```

で、80H から CMDLIN という名前で 128 バイト分を確保しています。これはコマンドラインのバッファが 128 バイトあるからです。

```
ORG 100H

MOV BL, CMDLN ［0］
CMP BL, 0
JE ERROR
```

で、BL には CMDLN のはじめの値（ファイル名の文字数）が入ります。それが、0 であれば、ファイル名の入力がなかったということでエラー処理にジャンプさせます。

```
MOV BH, 0
MOV CMDLN ［BX+1］, 0
```

で、BX にはファイル名の文字数が入り、それに +1 したアドレスがファイル名の終わりですから、そこに 0 を書き込みます（ASCIZ）。

```
MOV DX, OFFSET CMDLN ［2］
MOV CX, 0
INT 21 H
JC ERROR
MOV BX, AX
```

で、DX にファイル名の先頭アドレス（CMDLN+2）がセットされ、ファイルが作成できます。

■ ファイルへの書き込み

ファイル（シーケンシャルファイル）への書き込み読み方は次のとおりです。

●ファイルへの書き込みのファンクション・コール。

```
実行：
    AH ← 40H
    DX ←書き込むバッファの先頭アドレス
    CX ←書き込むバイト数
    BX ←ファイルハンドルの値
    INT 21 H
結果：
    CF=0 で正常終了。AX に書き込まれたバイト数が入る。
    CF=1のときはエラー。
```

1バイト「A」を書き込むプログラムは、次のようになります。

```
    MOV AH, 40H
    MOV DX, OFFSET BUFF
    MOV CX, 1
    INT 21 H
    JC ERROR

BUFF DB 'A'
```

このように、書き込みは、ユーザーがバッファを決め、バイト数を決めて行います。なお、BX にはファイルハンドルが入っていることを常に確認しておく必要があります。

ファイルのクローズ

●ファイル・クローズのファンクション・コール。

ファイルのクローズのしかたは前出と同じで、次のとおりです。

```
実行：
    AH ← 3EH
    BX ←ファイルハンドルの値
    INT 21 H
結果：
    CF=0 で正常クローズされた。CF=1のときはエラー。
```

このプログラム例は、次のようになります。

MOV AH, 3EH
INT 21H

BX にはファイルハンドルが入っていることを確認。

■ファイル作成プログラム

これまでの知識をもとに、練習としてファイル作成プログラムを作ってみましょう。これは、次のような機能を持つものです。

- キーボードから入力した文字をファイルに書き込む。
- CTRL+Z が入力されたらファイル書き込みを終了する。

● CTRL+Z を押すとテキストファイルの終わりを示すコードである 1AH が入力される。

次に、そのリストを挙げます。

■ファイル作成プログラム

```
;---------------------------------------
; FMAKE.ASM
; File Make Program
; Usage: FMAKE filename
;---------------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG 80H
CMDLN   DB 128 DUP (?) ;Command line buffer ―― ファイル名が格納されるエリア
;
        ORG 100H
;---------------------------------------
START:
CHECK:   MOV BL,CMDLN[0]
         CMP BL,0        ; If no filename   ┐ ファイル名がなければ
         JE ERROR        ; then ERROR       ┘ エラー処理へ
         MOV BH,0                           ┐ ファイル名の最後に
         MOV CMDLN[BX+1],0 ; ASCIZ for      ┘ 0を付ける
                           ; File Open
FCREATE:
        MOV AH,3CH         ; File Create    ┐
        MOV DX,OFFSET CMDLN[2]              │ ファイルの作成
        MOV CX,0           ; File Attrib    │
        INT 21H                             ┘
        JC ERROR          ; Error then ERROR ―― エラー処理
        MOV BX,AX         ; BX <= File Handle ―― BXにファイルハンドルを入れる
;-----------------------------------------
INPUT:
        MOV AH,1          ; INPUT AL        ┐ キーボードから
        INT 21H                             ┘ 1文字入力する
        MOV BUFF,AL       ; BUFF <- AL ―― 入力した1文字をBUFFに入れる
;-----------------------------------------
        CALL FWRITE       ; Write ―― ファイル書き込みルーチンをコール
;-----------------------------------------
```

```
CHR_CHK:
        CMP BUFF,1AH       ; If End Character
        JE FCLOSE          ; then Write end
        CMP BUFF,13
        JNE INPUT          ; Input again
        MOV AH,2           ; Print Line Feed
        MOV DL,10
        INT 21H
        MOV BUFF,10        ; Store Line Feed
;-------------------------------------------
        CALL FWRITE        ; Write
        JMP SHORT INPUT; Input again
;-------------------------------------------
FCLOSE: 
        MOV AH,3EH         ; File close
        INT 21H
;---------------------------------------
DOS:
        MOV AH,0
        INT 21H              ; Return to DOS
;=======================================
FWRITE:                      ; File Write SUB
        MOV AH,40H
        MOV DX,OFFSET BUFF
        MOV CX,1             ; Write one byte
        INT 21H
        RET                  ; RETURN
;=======================================
ERROR:
        MOV DX,OFFSET ERROR_MSG
        MOV AH,9             ; Print Message
        INT 21H
        JMP SHORT DOS        ; Return to DOS
;---------------------------------------
ERROR_MSG DB "File can't create.$"
BUFF       DB 1 (?)
CODE    ENDS
        END START
```

1AH(CTRL＋Z)が入力されたら終了

CR(キャリッジリターン)でなければ入力を続ける

CRであればLF(改行)する

LFコードをBUFFに入れる

ファイル書き込みルーチンをコール

入力をくりかえす

ファイル・クローズ

MS-DOSへ戻る

ファイルに1バイト書き込むサブルーチン

エラーメッセージ表示

DOSに戻る

エラーメッセージ

入力された1文字がここに入り，ファイルに書き込まれる

プログラムのポイントは、次のとおりです。

- ファイル名がなければ、エラーメッセージを表示して終了。
- 読み込みバッファ（BUFF）を1バイト確保して、1バイトずつキーボードから入力したものをファイルに書いている。

MOV AH, 1 ；1 文字入力
INT 21

・MOV BUFF, AL で直接、レジスタからメモリに内容を入れることができる。
・書き込んだ1バイトがファイルの終わりを示す1AHならば、書き込みを終了する。

次に実行例を挙げます。

■ファイル作成実行例

```
A>FMAKE DEMO ⏎
This is a demo. ⏎
This is a test. ⏎
This is a sample file.⏎
CTRL + Z を押すと，ファイルが作られる
```

●作成したファイルの内容を確認

```
A>TYPE DEMO ⏎
This is a demo.
This is a test.
This is a sample file.

A>
```

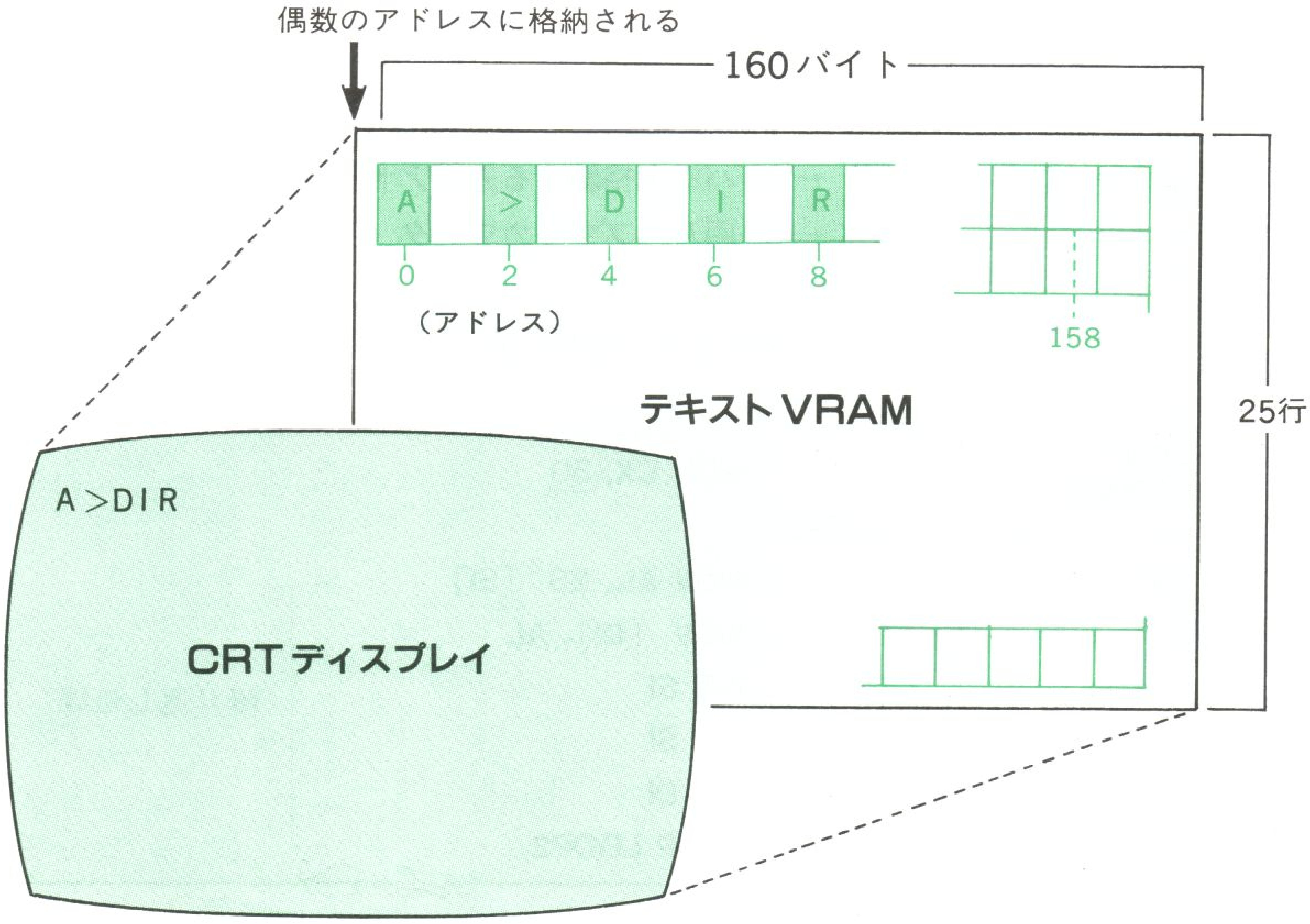
ちゃんとデータが入っている



■ 画面情報の取り出し

98のテキスト画面で、1バイトの英数字が表示された状態は、次のようになっています。

■画面表示とテキストVRAMとの関係



・偶数のアドレスだけに文字が格納されている。
・奇数のアドレスは、漢字（2バイト文字）表示用である。

そこで、画面情報の取り出しは、次のように行います。

・テキスト画面のセグメント・アドレス0A000H、オフセット・アドレス0から1バイトおきに内容を取り出す。
・1行80文字を取り出して、そのあとに復帰・改行コード（13. 10）を付ける。これは、1行表示したときに改行させるため。
・この82バイト分をファイルに書き込む。
・これを25行分繰り返す。

■ 2重ループにチャレンジ

横に80文字、縦に25行となると2次元の世界になります。この繰り返し処理をするには、ループを2重にしなくてはなりません。BASIC言語でもFOR-NEXTの2重を使うことがありますが、考え方はそれと同じです。

まず1つめのループ、横80文字の取り出しからプログラミングします。各レジスタは、次のように割り当てます。

- ES ←テキト画面のセグメント・アドレス
- SI ←そのオフセット・アドレス
- DI ←80バイト格納する先頭アドレス
- CX ←80回ループするカウンタ

すると、次のようになります。

●80回の繰り返し処理をする1重ループ。BASIC言語では次のようになる。

```
FOR CX=1 TO 80
  繰り返し処理
NEXT CX
```

```
        MOV CX, 80
LOOP2 :
        MOV AL, ES : [SI]    ┐
        MOV  [DI], AL        │
        INC SI               │
        INC SI               │ 繰り返し処理
        INC DI               │
        LOOP LOOP2           ┘
```

このループは、すでに学んだLOOP命令を使います。ポイントは、次のとおりです。

＜テキスト画面からデータを取るとき、＞

MOV AL, ES : [SI]

のように、[SI]の前にES：を付ける。

●ES:は、次にくるレジスタを修飾するもので、セグメントをESに設定する命令。ES:[SI]は、SIが指すアドレスの内容は、ESが指すセグメントにしている。DSが指すセグメントではない。

＜自分のプログラム内のメモリにデータを格納するには、＞

MOV [DI], AL

とする。これは、実は

MOV DS：[Di], AL

ということで、DS の場合は DS：は省略できるため、普通は DS：を付けないのです。

このように、2つのセグメントを取り扱うときのために、ES が用意されています。

<SI のアドレスは、1バイトおきなので2つずつ増やしている。>

▶ 2つめのループ

レジスタは、次のように割り当てます。

<BX ← 25 回ループのカウンタ>

そして、次のようにはじめのループをはさみます。

```
         MOV BX, 25
              ・
              ・
              ・
LOOP1：  MOV CX, 80
LOOP2：
          80 回ループ
          LOOP LOOP2
          DEC BX
          JNZ LOOP1；25 回ループ
```

JNZ

●80 回の繰り返し処理をするループを 25 回繰り返す2重ループになる。BASIC 言語では次のようになる。

```
FOR BX=1 TO 25
    ・
    ・
    ・
FOR CX=1 TO 80
  80 回ループ
NEXT CX
NEXT BX
```

これで、80×25 の 2,000 回の繰り返しができます。

この動きを説明します。

- まず、BX に 25 がセットされる。
- 次に、CX に 80 がセットされる。
- LOOP LOOP2 で CX が 0 になるまで LOOP2 のところを繰り返す。80 回ループする。
- BX から1を引いて、BX が 0 でなければ LOOP1 に飛ぶ。
- CX に 80 がセットされ、CX が 0 になるまで LOOP2 のところを繰り返す。80 回ループする。

・BX から 1 を引いて、BX が 0 になれば 25 回のループが終了する。

■画面表示保存プログラム

これで、やっと画面表示保存プログラムを作ることができます。次に、プログラムを挙げます。

■画面表示保存プログラム

```
;---------------------------------------
; SAVE.ASM
; Screen Save for ASCII characters
; Usage: SAVE filename
;---------------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG 80H
CMDLN   DB 128 DUP (?) ;Command line buffer   ── ファイル名が格納されるエリア
;
        ORG 100H
;---------------------------------------
START:
CHECK:   MOV BL,CMDLN[0]                      ┐
         CMP BL,0          ; If no filename    ├ ファイル名がなかったらエラー処理へ
         JE ERROR          ; then ERROR        ┘
         MOV BH,0                              ┐
         MOV CMDLN[BX+1],0 ; ASCIZ for         ├ ファイル名の最後に0を付ける
                           ; File Open         ┘
FCREATE:
        MOV AH,3CH          ; File Create      ┐
        MOV DX,OFFSET CMDLN[2]                 │
        MOV CX,0            ; File Attrib      ├ ファイル作成
        INT 21H                                ┘
        JC ERROR          ; Error then ERROR  ── エラー処理
        MOV FILE,AX       ; FILE <= File Handle ── ファイルハンドルをFILEに格納
;-----------------------------------------
GET_VRAM:
        MOV AX,0A000H                          ┐
        MOV ES,AX          ; ES <= 0A000H      ┘ テキスト画面のセグメントをESにセット
        MOV SI,0           ; VRAM Start addr   ┐
        MOV BX,25          ; 25 lines          ├ カウンタをセット
        MOV DI,OFFSET BUFF                     ┘
LOOP1:                     ; Data buffer
        MOV CX,80          ; Loop 80 times
LOOP2:
        MOV AL,ES:[SI]     ; Read one byte   ── 画面から1バイトとる
        MOV [DI],AL        ; Store one byte  ── それをバッファに入れる
        INC SI             ; Skip 2 bytes      ┐
        INC SI             ;                   ┘ アドレスを2つずらす
        INC DI             ; Inc store ad    ── アドレスを1つずらす
        LOOP LOOP2         ; Repeat 80 times ── これを80回くり返す
```

```
;----------------------------------------
        CALL FWRITE       ; 80 bytes write         ファイルに80バイト分書き込む
        MOV DI,OFFSET BUFF ; Reset Buffer ad        DIをセットしなおす
        DEC BX            ; Dec line counter
        JNZ LOOP1         ; If BX<> then LOOP1     25回ループ
;----------------------------------------
FCLOSE:
        MOV AH,3EH      ; File close
        MOV BX,FILE     ; File handle              ファイル・クローズ
        INT 21H
;------------------------------------
DOS:
        MOV AH,0                                   MS-DOSに戻る
        INT 21H           ; Return to DOS
;------------------------------------
ERROR:
        MOV DX,OFFSET ERROR_MSG
        MOV AH,9          ; Print Message          エラーメッセージ表示
        INT 21H
        JMP SHORT DOS     ; Return to DOS
;====================================
FWRITE:
        PUSH BX           ; Save BX,CX
        PUSH CX
        MOV AH,40H        ; File Write
        MOV DX,OFFSET BUFF
        MOV CX,82         ; Write 80 bytes+CRLF
        MOV BX,FILE       ; BX <= File handle
        INT 21H
        POP CX            ; Restore CX,BX
        POP BX
        RET
;====================================
ERROR_MSG DB "File can't create.$"
FILE       DW 1 (?)
BUFF       DB 80 DUP (?)
CRLF       DB 13,10
CODE    ENDS
        END START
```

プログラムのポイントは、次のとおりです。

•BX レジスタをカウンタとして使っているので、ファイルへの書き込みのときは、メモリ（FILE で定義）に格納したファイルハンドルの値を一時、BX に入れている。

MOV FILE, AX ；ファイルハンドルを保存
MOV BX, FILE ；ファイルハンドルの取り出し

● MOV FILE,AX はレジスタの値をメモリに直接書き込む例で、FILE の指すアドレスに AX の内容が入る。

●MOV BX,FILE はメモリの内容をレジスタに読み込む例で、FILE の指すアドレスの内容が BX に入る。

FILE の定義は、AX や BX とメモリとのやり取りなので、1ワード（16ビットで2バイト）としている。

FILE DW 1（?）　;　Define Word

DB では、アセンブルのときにエラーになるので注意。

・テキスト画面のデータ取り出しは、ES を使っている。

MOV AL, ES：[SI]

・ファイルへの書き込みは、わかりやすいようにサブルーチンにしている。

CALL FWRITE

・そのサブルーチンでは、ループカウンタが入っているレジスタ BX, CX を使うので、カウンタの値が壊れないように保存している。

```
PUSH BX
PUSH CX
   ・
   ・
POP CX
POP BX
```

・PUSH・POP では、一番最後に PUSH したものから順に POP しないと、ちゃんともとの値に戻らないので要注意。

```
PUSH BX
PUSH CX
   ・
   ・
POP BX
POP CX
```

とすると、BXとCXの値が入れ替わることになります。

●PUSHしたものは、必ずPOPしないとプログラムが暴走するので要注意。

・ファイルへの書き込みでは、バッファ80バイト分に復・改行コードを加えて82バイトを書いている。

```
MOV CX, 82
```

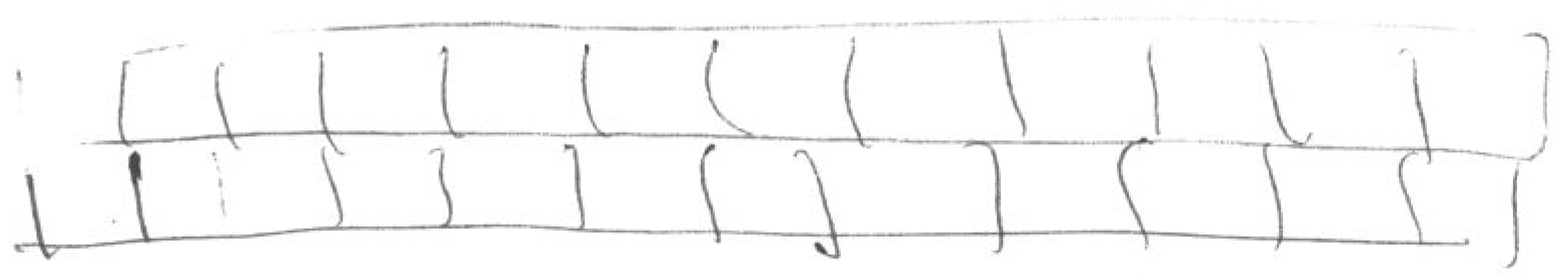
・バッファには、80バイト確保して、そのあとに13.10を格納している。

```
BUFF DB 80 DUP (?)
CRLF DB 13, 10
```

・82バイトを書いたあと、BXの値を−1する前に、DIで示すバッファのアドレスセットし直している。

```
MOV DI, OFFSET BUFF
DEC BX
JNZ LOOP1
```

こうしないと、バッファのアドレスがずれていくことになります。



3・3 INTコールあれこれ

第2部1章で、ソフトウェア割り込みの説明をしましたが、ここではそのINTコールを使った、簡単でちょっと面白いプログラムをいくつか紹介します。

これらは、MS-DOSのINT 21Hのコールとは違った98独自のものですが、MS-DOS上からコールすることができます。これにより、8086のキーポイントであるINTコールの理解がより深まることと思います。

98専用のINTコール

これまで、MS-DOSのINTコール（ファンクション・コール）をいろいろと使ってきましたが、98専用のINTコールも用意されており、そのうちのいくつかは簡単にユーザーが利用することができます。

ここで紹介する98専用のINTコールは、次のとおりです。

▶テキスト画面コピー

```
INT 5
```

これだけで、COPYキーを押したのと同じ働きをします。プログラム中で画面コピーをとることができます。

▶テキスト画面クリア

```
MOV DL, 0
MOV DH, 0E1H
INT 18H
```

DLに文字を入れると、それで画面全体が埋まります。

DHに特別な値を入れると、画面の色が変わります。

プログラム中でいろいろな画面クリアができます。

▶ビープ音を鳴らす・止める

<ビープ音を鳴らす>

```
MOV AH, 17H
INT 18H
```

<ビープ音を止める>

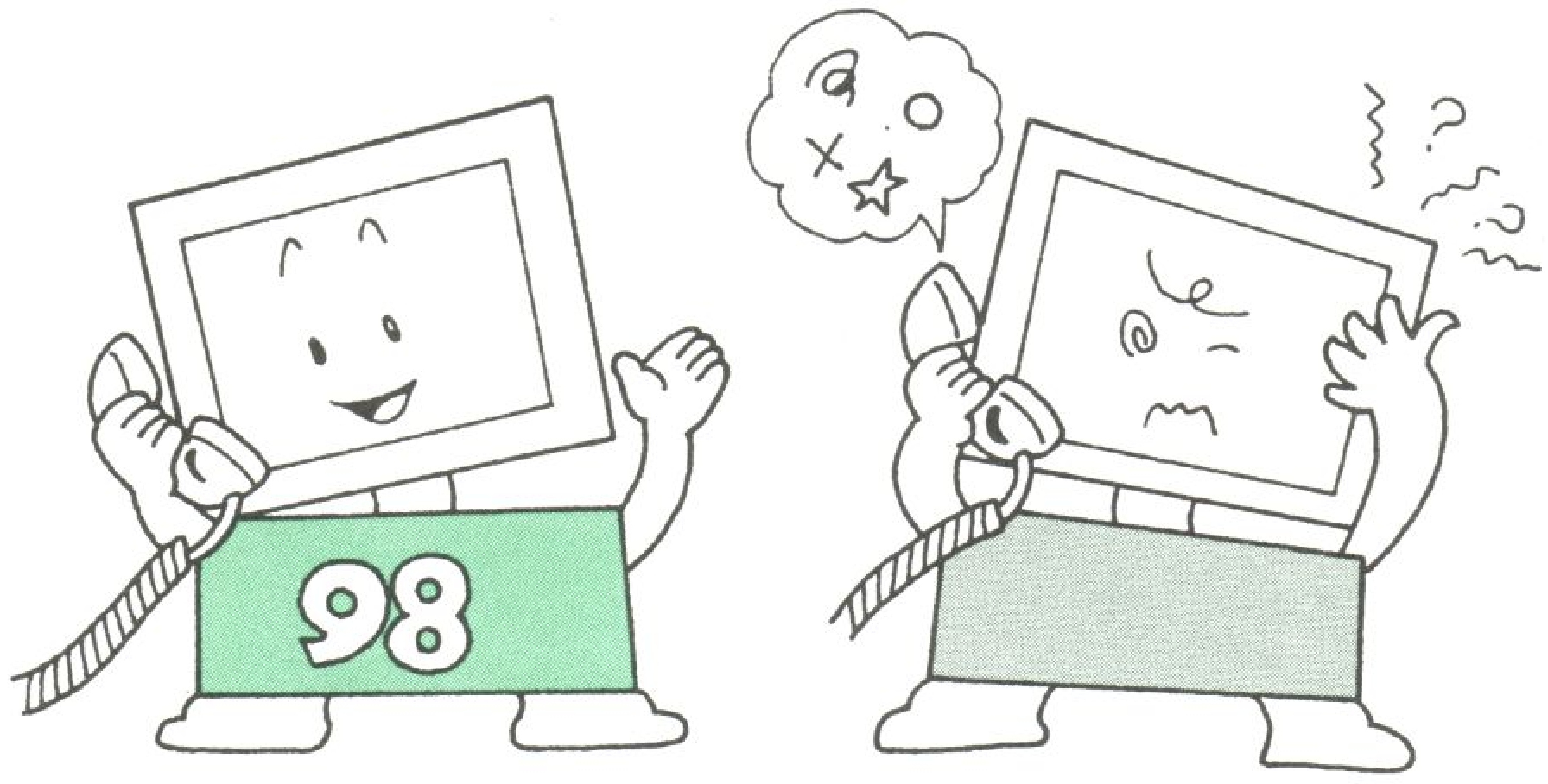
```
MOV AH, 18H
INT 18H
```

MS-DOSのファンクション・コールを使うよりも簡単です。

注意事項：

- これらのINTコールは、98専用ですので、98以外のマシンでは使えません。
- MS-DOSのプログラムとしては、他のマシンとの互換性が失われます。
- こういったINTコールは、メーカーからは公開されていないので書籍や雑誌で紹介されているのを利用するか、自分で解析して見つけることになります。

次に、これらの**INT**を使ったプログラムをお目にかけます。



98専用のINTコール

■テキスト画面コピープログラム

■テキスト画面コピープログラム

```
;--------------------------
;   SCOPY.ASM
; Screen Copy
;--------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
START:
     INT 5        ; COPY key routine
     MOV AH,0
     INT 21H      ; Return to DOS
;-----------------------------------
CODE ENDS
     END START
```

INT 5を実行するだけ

まるで簡単なものです。本書で紹介する1番短いプログラムです。それでも、実用になります。

■画面コピー実行例

```
A>DIR *.BAT

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

FDISP    BAT      413  87-07-23   1:01
MENU     BAT       50  87-07-02  23:09
DIRSORT  BAT      547  87-07-14   0:00
CONSET   BAT      785  87-07-04  21:13
FILES    BAT      657  87-07-09   1:59
PARA     BAT      112  87-07-09   2:01
FTYPE    BAT      427  87-07-13  23:26
TFILES   BAT       50  87-07-09  14:57
MNU      BAT     1130  87-07-23   2:18
ASM      BAT       64  87-05-17  20:08
       10 個のファイルがあります.
   576512 バイトが使用可能です.
A>SCOPY
```

画面表示

画面コピープログラムを実行する

■テキスト画面クリア・プログラム

■テキスト画面クリア・プログラム

```
;----------------------------------
; CLEAR.ASM
; Screen Clear
;----------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG   100H
;---------------------------------------
START:
        MOV AH,16H    ; Clear Command
        MOV DL,0      ; Text Clear
        MOV DH,0E1H   ; Attrib:Normal B/W
        INT 18H       ; Clear
;----------------------------------------
DOS:
        MOV AH,0
        INT 21H           ; Return to DOS
;----------------------------------------
CODE    ENDS
        END START
```

普通のクリア (MOV AH,16H ～ INT 18H)

DL には、クリアする文字を入れます。

DH には、画面を修飾するアトリビュート(属性)を入れます。白黒でノーマル表示は、0E1Hです。次の例は、文字をX、色を黄色にしてクリアするものです。

■「X」で画面を埋め黄色にする例

```
;--------------------------------
; CLEAR2.ASM
; Screen Clear
;--------------------------------
CODE    SEGMENT
        ASSUME CS:CODE,DS:CODE,ES:CODE
        ORG   100H
;--------------------------------------
START:
        MOV AH,16H    ; Clear Command
        MOV DL,'X'    ; Text Clear
        MOV DH,0C1H   ; Attrib Yellow
        INT 18H       ; Clear
;---------------------------------------
DOS:
        MOV AH,0
        INT 21H           ; Return to DOS
;---------------------------------------
CODE    ENDS
        END START
```

「X」を表示 (MOV DL,'X')

黄色にする (MOV DH,0C1H)

■「X」を表示し、黄色にした例

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
A>
```



■SOS発生プログラム

■SOS発生プログラム

```
;-------------------------
;  SOS.ASM
; SOS sound generator
;-------------------------
COUNTER EQU 20000
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
START:
     MOV SI,OFFSET SOS
REPEAT:
     MOV AL,[SI]      ; AL <= CHR
     INC SI
     CMP AL,0         ; If 0 then End
     JE DOS
;-------------------------------------
     CMP AL,' '       ; If SPACE
     JE OFF           ; then BEEP OFF
ON:
     MOV AH,17H       ; If - then
     INT 18H          ; BEEP ON
     CALL TIMER       ; Wait
     JMP SHORT REPEAT ; Get another CHR
OFF:
     MOV AH,18H
     INT 18H          ; BEEP OFF
     CALL TIMER       ; Wait
     JMP SHORT REPEAT ; Get another CHR
DOS:
     MOV AH,18H
     INT 18H          ; BEEP OFF
     MOV AH,0
     INT 21H          ; Return to DOS
;-------------------------------------
TIMER:
     MOV CX,COUNTER ; Set Counter
WAIT:
     LOOP WAIT        ; Wait loop
     RET              ; RETURN
;-------------------------------------
SOS  DB '- - -   --- --- ---   - - -',0
CODE ENDS
     END START
```

- COUNTER EQU 20000：この数を変えると音色が変わる。
- MOV AL,[SI] / INC SI：データを1つずつとってくる
- CMP AL,0 / JE DOS：データが0なら終了
- CMP AL,' ' / JE OFF：スペースであれば，ビープ音を止める そうでなければ鳴らす
- ON：ビープ音を鳴らす
- OFF / DOS：ビープ音を止める
- TIMER：時間待ちルーチン
- SOS DB：モールス信号データ

DEC CX
~~LOOP~~ WAIT
JNZ

●マシン語での処理は、実行速度が速いので、ときに時間待ちをする必要がある。時間待ちは、ちょっとしたタイミングをとるときに有用。ここでは、ビープ音を鳴らしたり止めたりする長さを調節している。

ビープ音とタイマーを使って、モールス信号でSOSを鳴らしています。−のときはオン、スペースのときはオフとしています。また、LOOP命令を使って、時間待ち(タイマー)をしています。これは、ただCXが0になるまでカウントダウンするだけで、ループの中身は何もありません。このようにアセンブリ言語では、ちょっとしたタイミングをとるときに、ダミーのループを使います。これは、覚えておくと役に立ちます。BASIC言語でも、

FOR I=1 TO 3000 : NEXT

と時間待ちをすることがありますが、同じです。

■独自のINTコールの方法

これまでは、すでにシステム側でプログラミングされているINTコールを使いましたが、ユーザー自身でも、独自のINTコールを作り、利用することができます。

それは、システム側で使っていないINTの番号があるからです。試しに、デバッガを起動して、次のようにしてみてください。

■INTコールの未使用アドレス

```
A>SYMDEB
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-D0:300,3FF
0000:0300  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0310  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0320  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0330  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0340  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0350  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0360  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0370  64 4E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   dN`.~^`.~^`.~^`.
0000:0380  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:0390  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:03A0  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:03B0  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:03C0  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:03D0  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:03E0  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
0000:03F0  7E 5E 60 00 7E 5E 60 00-7E 5E 60 00 7E 5E 60 00   ~^`.~^`.~^`.~^`.
-Q
A>
```

INT DCH（ファンクションキー読み書き） ← 0370 64 4E 60 00

INT 0FFH ← 3FC(7E) 3FD(5E) 3FE(60) 3FF(00)

「7E5E 6000」のところは，すべて未使用

これは、INT コールの飛び先が格納されている部分の終わりのほうです。アドレス 370H から 4 バイトをみると、

64 4E 60 00

となっていますが、これは実は、ファンクションキーの読み書きをした INC 0DCH（セグメント 0060H, オフセット 4E64H に処理が移る）です。

これ以外は、すべて

7E 5E 60 00

となっており、この部分は INT コールが未定義で、使われていないのです。未定義 INT コールを実行すると、ビープ音が鳴り、

Int trap halt

と表示されて、システムがストップしてしまいます。これは、未定義 INT コールが実行されたので、それ（Int）を捕らえて（trap）、ストップ（halt）したという意味です。

このプログラムと実行例は、次のようになります。

■INT OFFHプログラム

```
;-------------------------
;  INTFF.ASM
; Test for INT call
;-------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
START:
;-----------------------------------
     INT 0FFH       ; INT call
;-----------------------------------
CODE ENDS
     END START
```

■INT 0FFH 実行例

```
A>INTFF ⏎ ———— ビープ音が鳴ってハングアップする

Int trap halt ——— 許されていないINT命令を実行
                  したメッセージ
```

未定義だから使えないのであって、定義すれば使えるようになります。INT 0FFH のところをみてください (P233)。

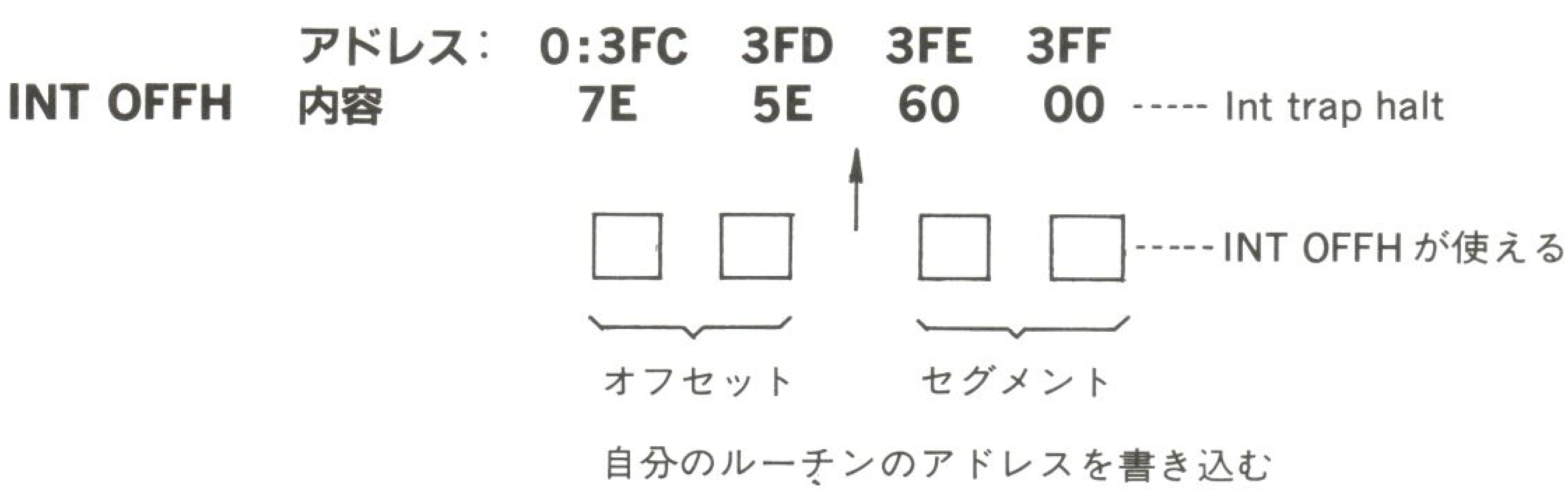
セグメント 0060H, オフセット 5E7EH の代わりに、別のアドレスを格納し、そのアドレスに実行したいルーチンを入れておけば、

INT 0FFH

で、自分のルーチンが実行できます。なお、自分のルーチンの最後には、INT コールから戻る命令 IRET (Interrupt RETurn) を書きます。すでに定義されている INT コールのルーチンの最後には、必ず IRET があります。

このアドレス書き換えを図示すると、次のようになります。

■INTの飛び先を変える



この節の冒頭で、INT 5 の画面コピーを実行しましたが INT 5 の飛び先を INT 0FFH の飛び先にすることにより、INT 0FFH で画面コピーがとれるようになります。この例は、自分専用のプログラムを作ると複雑になるため、すでに用意されているルーチンを利用して、未定義の INT コールが使えるようにしようというわけです。

プログラム例：

■INT OFFHをINT 5と同じにする

```
;----------------------------
;   SCOPY2.ASM
;  Screen Copy with INT 0FFH
;----------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
START:
     MOV AX,0
     MOV DS,AX
     MOV SI,14H        ; INT 5 addr      ―― INT5のアドレス
     MOV DI,3FCH       ; INT 0FF addr    ―― INT OFFHのアドレス
;-------------------------------------------
     CLI               ; Clear INT       ―― インタラプトを禁止する
     MOV AX,[SI]
     MOV BX,[SI+2]     ; Load INT 5 addr ―― INT5のアドレス
     MOV [DI],AX                            ↓
     MOV [DI+2],BX     ; Store INT 0FFH  ―― INT OFFHのアドレス
     STI               ; Set INT         ―― インタラプトを許可する
;-------------------------------------------
     INT 0FFH       ; COPY key routine   ―― INT OFFHが使える
     MOV AH,0
     INT 21H        ; Return to DOS
;-------------------------------------------
CODE ENDS
     END START
```

INT 5の飛び先は、セグメント0, オフセット14～17Hに格納されていますので、INT 0FFHのアドレス(3FCH–3FFH)に転送しています。

転送の前後のCLIとSTIの意味は、次のとおりです。

CLI (CLear Interrput flag)

●CLIは、割り込みを禁止する命令。通常は、割り込みができる状態になっている。

割り込みを禁止する命令で、これが実行されると以降は、割り込みができなくなります。インタラプト・テーブルの内容を書き変えるときは、いつ、そこに割り込みがかかって飛んでくるかも知れませんので、割り込みを禁止しているわけです。この場合は、未定義の部分ですのでここに飛んでくる恐れはないのですが、万一のことを考えて割り込み禁止にしています。定義されている部分を書き変えるとき、割り込み禁止にしておかないと、暴走することがあります。

●STIは、割り込みを許可する命令。

STI（SeT Interrput flag）

割り込みを許可する命令で、もとどおりいつでも割り込みがかかる状態になります。

実行例：

■INT OFFHでの画面コピー

```
A>DIR *.EXE ⏎

 ドライブ A: のディスクのボリュームラベルはありません.
 ディレクトリは A:¥

MASM       EXE     77362  86-10-13    0:00
CREF       EXE     10544  86-10-13    0:00
LINK       EXE     41114  86-10-13    0:00
LIB        EXE     24138  86-10-13    0:00
MAKE       EXE     18675  86-10-13    0:00
MAPSYM     EXE     51904  86-10-13    0:00
SYMDEB     EXE     36538  86-10-13    0:00
VE         EXE     45926  85-03-19   14:21
EXE2BIN    EXE      2880  86-10-13    0:00
CHKDSK     EXE      9760  86-10-13    0:00
FIND       EXE      6525  86-10-13    0:00
SORT       EXE      1680  86-10-13    0:00
EDLIN      EXE      7426  86-10-13    0:00
N88BASIC   EXE     98816  84-11-14   17:47
       14 個のファイルがあります.
   544768 バイトが使用可能です.
A>SCOPY2 ⏎
```

画面表示

INT OFFHで画面コピー

●ATOK6も独自のINTコールを使っている。

INT 6FH
と
いい番組

ちなみに、98用の日本語フロントプロセッサATOK6はINT 6FHを使い、かな漢字変換の入力モードの切り替えなどを行っています。

■リセット・プログラム

では、独自のINTコールを使って98をリセットするプログラムを作りましょう。これは、リセット・ボタンを手で押すのと同じ働きをするものです。そのメリットは、プログラム中で、自動的にリセットがかけられることですが、次のような場合に役に立ちます。

・MS-DOS のシステムから自動的に別のシステム（DISK BASIC など）を起動する。
・MS-DOS のシステムをリセットして、別の CONFIG. SYS の内容を読み込む（ATOK6 を使っていて VJE-β に切り替えたいなど、これは特にハードディスクで利用すると自動的に使いたいシステムが起動できる）。

リセット・プログラムを作るには、8086 にリセットがかかると、どこから実行を始めるかを知っておく必要があります。それは、電源 ON をしたときと同じですが、次のアドレスから実行を始めます。

セグメント 0FFFFH
オフセット 0000H

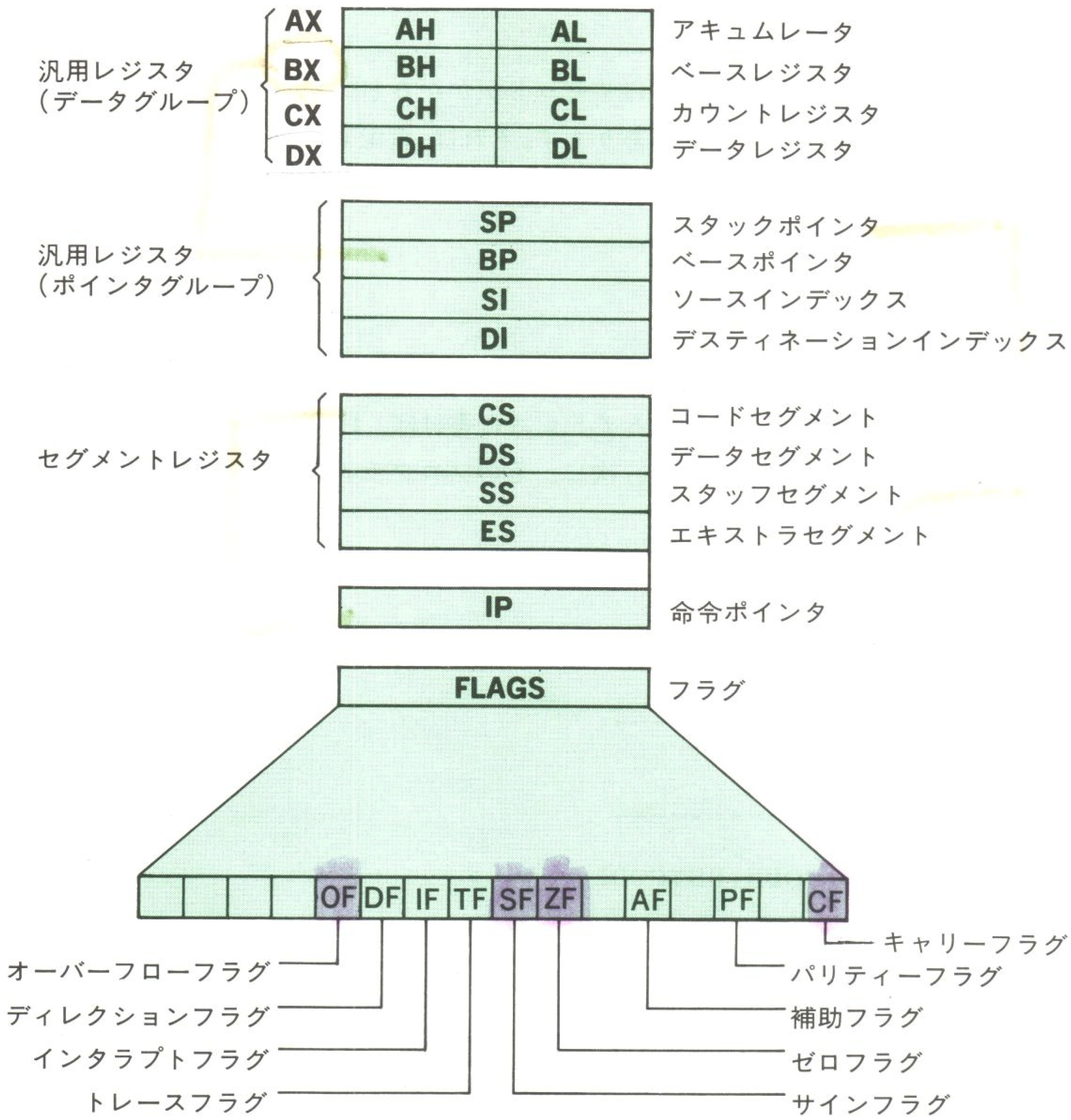
これを、INT 0FFH の飛び先を格納するアドレスに書けば、INT 0FFHでリセットがかかるというわけです。次に、そのプログラムを挙げます。

●システムをリセットするからといって、本書で学んだことを頭の中でリセットしないでください。本書でのプログラミングをベースにしてさらに高度な技法を身につけられることを願っています。

■リセット・プログラム

```
;-------------------------
;  RESET.ASM
; Reset MS-DOS
;-------------------------
CODE SEGMENT
     ASSUME CS:CODE,DS:CODE
     ORG 100H
START:
     MOV AX,0     ; DS <= 0
     MOV DS,AX
     MOV BX,3FCH    ; INT 0FFH addr.────── INT 0FFHのアドレス
     MOV AX,0       ; offset  0000H
     MOV CX,0FFFFH ; Segment 0FFFFH
;-----------------------------------
     CLI            ; Clear INT ────────── インタラプトを禁止する
     MOV [BX],AX    ; 3FCH <= 0000H   ┐
     MOV [BX+2],CX ; 3FEH <=0FFFFH   ┘リセットするアドレスを入れる
     STI            ; Set INT ──────────── インタラプトを許可する
     INT 0FFH       ; Reset ────────────── INT 0FFHの実行
;-----------------------------------
CODE ENDS
     END START
```

付録1　8086のレジスタ一覧



付録2　ニーモニックの原語と意味

（本書で学んだもののみ）

◉データ転送

ニーモニック	英　　語	意　　味
MOV	ムーブ MOVe	転送(ロード)する
PUSH	プッシュ　データ PUSH data	押さえ込む(退避)
POP	ポップ　データ POP data	引き上げる(復帰)

◉加　算

ニーモニック	英　　語	意　　味
ADD	アッド ADD	加える
INC	インクリメント INCrement	増やす(I を加える)

◉減　算

ニーモニック	英　　語	意　　味
SUB	サブトラクト SUBtract	引く(減算)
DEC	デクリメント DECrement	減少(I を引く)
CMP	コンペア CoMPare	比較(引き算をして比較)

◉ストリング(一連のメモリ)操作

ニーモニック	英　　語	意　　味
MOVSB	ムーブ　ストリング　バイト MOVe String Byte	バイト転送
MOVSW	ムーブ　ストリング　ワード MOVe String Word	ワード転送
REP	リピート REPeat	くり返し

◉ジャンプ

ニーモニック	英　　語	意　　味
JMP	ジャンプ JuMP	ジャンプ(GOTO)
JE	ジャンプ オン イクオール Jump on Equal	等しければジャンプ
JNE	ジャンプ オン ノット イクオール Jump on Not Equal	等しくなければジャンプ
JNZ	ジャンプ オン ノット ゼロ Jump on Not Zero	ゼロでなければジャンプ
JA	ジャンプ オン アバブ Jump on Above	より大ならジャンプ
JAE	ジャンプ オン アバブ オア イクオール Jump on Above or Equal	より大か等しいならジャンプ
JB	ジャンプ オン ビロウ Jump on Below	より小ならジャンプ
JBE	ジャンプ オン ビロウ オア イクオール Jump on Below or Equal	より小か等しいならジャンプ
LOOP	ループ Loop	CX-1のあと CX＜＞0ならジャンプ

◉サブルーチン

ニーモニック	英　　語	意　　味
CALL	コール CALL	サブルーチンコール （セグメント内）
RET	リターン RETurn	リターン(セグメント内)

◉割り込みルーチン

ニーモニック	英　　語	意　　味
INT	インタラプト INTerrupt	割り込みルーチンコール
IRET	インタラプト リターン INTerrupt RETurn	割り込みルーチンからの リターン

◉フラグ制御

ニーモニック	英　　語	意　　味
CLD	クリア ディレクション CLear Direction	方向をプラスにする
STD	セット ディレクション SeT Direction	方向をマイナスにする
STI	セット インタラプト SeT Interrupt	割り込み許可
CLI	クリア インタラプト CLear Interrput	割り込み禁止

付録3　8086の基本命令一覧

ここに、本書で解説した８０８６の命令を機能別にまとめて
掲載します。すべて基本的で、最低限覚えておきたいものです。

1　データ転送命令

```
MOV　　転送先，転送元
```

転送元の内容を転送先に移す。転送元の内容は変わらない。

■セグメント・レジスタのアドレス転送

```
MOV AX,0A000H
MOV DS,AX
MOV ES,AX

MOV AX,CS
MOV DS,AX
MOV ES,AX
```

● DS と ES を A000H にする。DS,ES には直接、値を入れられないので AX を利用する。

● DS と ES を CS の値と同じにする。

■メモリ上のアドレス転送

```
MOV SI,OFFSET BUFF
MOV DI,OFFSET BUFF2
MOV DX,OFFSET MSG
MOV SI,OFFSET BUFF+160
```

●特定のメモリのアドレスをレジスタに入れるには、疑似命令 OFFSET を使う。アドレスの値を調節したいときは、+, −などが使える。

■レジスタで指定したアドレスの内容読み書き

● SIが示すアドレスから1バイトの内容をALに入れる。

● SIが示すアドレスにAHの内容を入れる。

● SIが示すアドレスから2バイトの内容をAXに入れる。

● SIが示すアドレスからBXの内容を入れる。

● SIが示すアドレス+1から1バイトの内容をALに入れる。

● CMDLNが示すアドレスの内容をBLに入れる。

```
MOV AL,[SI]
MOV AH,[DI]

MOV [SI],AH
MOV [DI],AL

MOV AX,[SI]
MOV BX,[DI]

MOV [SI],BX
MOV [DI],AX

MOV AL,[SI+1]
MOV [DI+1],AL

MOV BL,CMDLN[0]
MOV CMDLN[BX+1],0
MOV DX,OFFSET CMDLN[2]
```

■レジスタの内容退避・復帰

● PUSH・POPは、16ビットのデータを対象とするため、

PUSH AL
POP AL

など8ビットのデータを取り扱うことはできない。ALだけを退避・復帰したいときでも、

PUSH AX
POP AX

としなくてはならない。

PUSH　レジスタ

レジスタの内容を特別なメモリ(スタック)に退避する。

```
PUSH CS
PUSH BX
PUSH CX
```

POP　レジスタ

PUSHで退避したレジスタの内容をレジスタに戻す（復帰する)。

```
POP CX
POP BX
POP DS
```

● PUSHした順番とまったく逆の順でPOPしないと、レジスタの内容が正常に復帰できない。また、PUSHしたら必ずPOPしなければならない。

2　加減算命令

●加算

```
ADD A, B
```

AにBを加えて、Aとする（A＝A＋B）。

```
ADD BX,80

ADD SI,160

ADD BX,SI

ADD DI,SI
```

●結果は次のとおり。

BX＝BX＋80
SI＝SI＋160
BX＝BX＋SI
DI＝DI＋DI

```
INC A
```

Aに1を加えて、Aとする（A＝A＋1）。

```
INC BX

INC SI

INC DI
```

●結果は次のとおり。

BX＝BX＋1
SI＝SI＋1
DI＝DI＋1

●減算

```
SUB　A, B
```

AからBを引いて、Aとする（A＝A－B）。

●結果は次のとおり。

AL＝AL－30H
AX＝AX－80
BX＝BX－SI
DI＝DI－DI

```
SUB AL,30H
SUB AX,80
SUB BX,SI
SUB SI,DI
```

```
DEC　A
```

Aから1を引いて、Aとする（A＝A－1）。

●結果は次のとおり。

BX＝BX－1
SI＝SI－1
DI＝DI－1

```
DEC BX
DEC SI
DEC DI
```

```
CMP　A, B　（比較）
```

AからBを引いた結果をフラグに反映させる。実際に引き算するのではないためAの値は変わらない。フラグは、A，Bが等しいとき、ゼロフラグが立つ。

```
CMP AL,'Y'

CMP AL,'N'

CMP AH,'0'

CMP DL,0
```

● CMP で比較をしたあと、次のような命令で分岐する。

- JE(等しいとき)
- JA(大きいとき)
- JB(小さいとき)

比較の主体は左側であることに注意。

3　ストリング操作命令（ブロック転送命令）

```
MOVS
```

ひとかたまりのメモリの内容を転送する。

```
REP MOVSB

REP MOVSW
```

● MOVSB は 1 バイト転送で MOVSW は 2 バイト転送。

4　分岐命令

```
CALL　ラベル
```

ラベルから始まるサブルーチンを呼び出す。

```
CALL LPRINT_SUB

CALL FKEYS
```

● BASIC では、次のようになる。

```
GOSUB ＊LPRINT_SUB
GOSUB ＊FKEYS
```

RET

サブルーチンからもとに戻る。

●BASIC では RETURN のこと。

```
RET
```

JMP　飛び先

無条件に、飛び先に制御を移す。

●JUMP SHORT は、ショートジャンプといい+－128バイト以内の範囲でのジャンプ命令。

```
JMP START
JMP SHORT ERROR
```

J・・　飛び先

フラグの状態により飛び先に制御を移す。

●JE と JZ とは、ともにゼロフラグがゼロかどうかを判断するので、CMP 命令で等しいときにジャンプする。

```
JE LPRINT_END
JZ LPRINT_END
JNZ START
JAE INPUT
JBE INPUT
```

LOOP 飛び先

CXから1を引いてCXの値が0でなければ、飛び先に制御を移す。

```
LOOP START
```

●LOOP命令の働きは、次のとおり。

CX＝CX－1
IF CX＜＞0 THEN
GOTO START

INT 型

割り込み処理で型の番号で決められたルーチンに制御を移す。

```
INT 21H
INT 0DCH
```

●INT 21Hは、MS-DOSの一般的なファンクション・コール。INT 0DCHは、ファンクションキー読み書きのファンクション・コール。

IRET

INTコールから戻る。

```
IRET
```

5 フラグ設定命令

CLD

ブロック転送のとき、アドレスを＋の方向に設定する。

● MOVSB のとき、

SI＝SI＋1
DI＝DI＋1
となり、

MOVSW のとき、

SI＝SI＋2
DI＝DI＋2

となる。

```
CLD
```

STD

ブロック転送のとき、アドレスを－の方向に設定する。

● MOVSB のとき、

SI＝SI－1
DI＝DI－1

となり、

MOVSW のとき、
SI＝SI－2
DI＝DI－2

となる。

```
STD
```

CLI

割り込みを禁止する。

● INT の割り込みアドレスを変更するときなど、割り込みがかかってはいけないときに CLI を用いる。

```
CLI
```

STI

割り込みを許可する。

```
STI
```

付録4 MS-DOSファンクション・コール実例集

1　MS-DOSに戻る

```
DOS:
        MOV AH,0
        INT 21H
```

2　1文字画面表示

■空白を表示する

```
PRINT_CHR:
        MOV AH,2
        MOV DL,' '
        INT 21H
```

■ビープ音を鳴らす

```
BEEP:
        MOV AH,2
        MOV DL,7
        INT 21H
```

3　複数の文字を画面に表示

■１文字表示を使ったもの（最後は０）

```
PRINT_CHRS:
        MOV SI,OFFSET BUFF
PRINT:
        MOV AH,2
        MOV DL,[SI]
        CMP DL,0
        JE PRINT_END
        INC SI
        JMP SHORT PRINT
PRINT_END:
```

```
BUFF DB 7,'Input Error',0    ;  この内容を表示
```

■複数の文字表示を使ったもの（最後は＄）

```
PRINT_CRLF:
        MOV AH,9
        MOV DX,OFFSET CRLF
        INT 21H

CRLF DB 13,10,'$'  ;  この内容を表示


PRINT_ERROR:
        MOV AH,9
        MOV DX,OFFSET ERROR
        INT 21H

ERROR DB 'File not found.$'    ;  この内容を表示
```

4　キーボードから１文字入力

■押されたキーが「Ｙ」であれば、次に進むルーチン

```
INPUT:
      MOV AH,1          ; 入力されたキーを表示する
      INT 21H           ; 入力にはリターンキーを押す
      CMP AL,'Y'
      JNE INPUT


INKEY:
      MOV AH,7          ; 入力されたキーを表示しない
      INT 21H           ; 入力にはリターンキーは不要
      CMP AL,'Y'
      JNE INKEY
```

5　キーボードから複数の文字入力

■キーボードから最大１２８文字を入力する

```
LINEINPUT:
      MOV AH,0AH
      MOV DX,OFFSET BUFF
      INT 21H

BUFF    DB 128              ; 入力する最大の文字数
BUFF2   DB 128 DUP (?)      ; 入力した文字を格納するエリア
```

6　プリンタ１文字出力

■プリンタに復帰コードを出力する

```
LPRINT_CR:
      MOV AH,5
      MOV DL,13
      INT 21H
```

■プリンタに復帰・改行コードを出力する

```
LPRINT_CRLF:
        MOV AH,5
        MOV DL,13
        INT 21H
        MOV AH,5
        MOV DL,10
        INT 21H
```

7　プリンタ複数文字出力

```
LPRINT_STR:
        MOV SI,OFFSET BUFF
LPRINT:
        MOV DL,[SI]
        CMP DL,0
        JE LPRINT_END
        MOV AH,5
        INT 21H
        INC SI
        JMP SHORT LPRINT
LPRINT_END:

BUFF DB '*** DATA ***',0    ; この内容を印字
```

8　ファイル作成

■ＴＥＳＴ．ＤＡＴを作成

```
FCREATE:
        MOV AH,3CH
        MOV DX,OFFSET FNAME
        MOV CX,0                    ; ファイル属性（通常）
        INT 21H
        MOV FILE_HANDLE

FNAME DB 'TEST.DAT',0               ; ファイル名
FILE_HANDLE DW 0                    ; ファイル・ハンドル格納エリア
```

9　ファイル・オープン

■ＴＥＳＴ．ＤＡＴを読み込みモードでオープン

```
FOPEN_READ:
      MOV AH,3DH
      MOV DX,OFFSET FNAME
      MOV AL,0                    ；ＡＬ＝１　書き込み
      INT 21H                     ；ＡＬ＝２　読み書き
      MOV FILE_HANDLE,AX

FNAME DB 'TEST.DAT',0             ；ファイル名
FILE_HANDLE DW 0                  ；ファイル・ハンドル格納エリア
```

10　ファイル・クローズ

■ファイル・ハンドルでオープンされているファイルのクローズ

```
FCLOSE:
      MOV AH,3EH
      MOV BX,FILE_HANDLE
      INT 21H
```

11　ファイルの読み出し

■１バイトの読み出し

```
FREAD:
      MOV HA,3FH
      MOV DX,OFFSET BUFF
      MOV CX,1                    ；　ＣＸ＝読み出すバイト数
      INT 21H

BUFF DB 0                         ；　読み出すエリア
```

12　ファイル書き込み

■１バイト書き込み

```
FWRITE:
      MOV AH,40H
      MOV DX,OFFSET BUFF
      MOV CX,1                  ;  ＣＸ＝書き込むバイト数
      MOV BX,FILE_HANDLE
      INT 21H

BUFF DB 13                      ;  書き込む内容
```

付録 5　10 進・16 進対応表

10 進	16 進	10 進	16 進	10 進	16 進
0 ···	00	32 ···	20	64 ···	40
1 ···	01	33 ···	21	65 ···	41
2 ···	02	34 ···	22	66 ···	42
3 ···	03	35 ···	23	67 ···	43
4 ···	04	36 ···	24	68 ···	44
5 ···	05	37 ···	25	69 ···	45
6 ···	06	38 ···	26	70 ···	46
7 ···	07	39 ···	27	71 ···	47
8 ···	08	40 ···	28	72 ···	48
9 ···	09	41 ···	29	73 ···	49
10 ···	0A	42 ···	2A	74 ···	4A
11 ···	0B	43 ···	2B	75 ···	4B
12 ···	0C	44 ···	2C	76 ···	4C
13 ···	0D	45 ···	2D	77 ···	4D
14 ···	0E	46 ···	2E	78 ···	4E
15 ···	0F	47 ···	2F	79 ···	4F
16 ···	10	48 ···	30	80 ···	50
17 ···	11	49 ···	31	81 ···	51
18 ···	12	50 ···	32	82 ···	52
19 ···	13	51 ···	33	83 ···	53
20 ···	14	52 ···	34	84 ···	54
21 ···	15	53 ···	35	85 ···	55
22 ···	16	54 ···	36	86 ···	56
23 ···	17	55 ···	37	87 ···	57
24 ···	18	56 ···	38	88 ···	58
25 ···	19	57 ···	39	89 ···	59
26 ···	1A	58 ···	3A	90 ···	5A
27 ···	1B	59 ···	3B	91 ···	5B
28 ···	1C	60 ···	3C	92 ···	5C
29 ···	1D	61 ···	3D	93 ···	5D
30 ···	1E	62 ···	3E	94 ···	5E
31 ···	1F	63 ···	3F	95 ···	5F

10進	16進	10進	16進	10進	16進
96 ···	60	128 ···	80	160 ···	A0
97 ···	61	129 ···	81	161 ···	A1
98 ···	62	130 ···	82	162 ···	A2
99 ···	63	131 ···	83	163 ···	A3
100 ···	64	132 ···	84	164 ···	A4
101 ···	65	133 ···	85	165 ···	A5
102 ···	66	134 ···	86	166 ···	A6
103 ···	67	135 ···	87	167 ···	A7
104 ···	68	136 ···	88	168 ···	A8
105 ···	69	137 ···	89	169 ···	A9
106 ···	6A	138 ···	8A	170 ···	AA
107 ···	6B	139 ···	8B	171 ···	AB
108 ···	6C	140 ···	8C	172 ···	AC
109 ···	6D	141 ···	8D	173 ···	AD
110 ···	6E	142 ···	8E	174 ···	AE
111 ···	6F	143 ···	8F	175 ···	AF
112 ···	70	144 ···	90	176 ···	B0
113 ···	71	145 ···	91	177 ···	B1
114 ···	72	146 ···	92	178 ···	B2
115 ···	73	147 ···	93	179 ···	B3
116 ···	74	148 ···	94	180 ···	B4
117 ···	75	149 ···	95	181 ···	B5
118 ···	76	150 ···	96	182 ···	B6
119 ···	77	151 ···	97	183 ···	B7
120 ···	78	152 ···	98	184 ···	B8
121 ···	79	153 ···	99	185 ···	B9
122 ···	7A	154 ···	9A	186 ···	BA
123 ···	7B	155 ···	9B	187 ···	BB
124 ···	7C	156 ···	9C	188 ···	BC
125 ···	7D	157 ···	9D	189 ···	BD
126 ···	7E	158 ···	9E	190 ···	BE
127 ···	7F	159 ···	9F	191 ···	BF

10進	16進	10進	16進	10進	16進
192 ···	C0	224 ···	E0	256 ···	100
193 ···	C1	225 ···	E1	512 ···	200
194 ···	C2	226 ···	E2	768 ···	300
195 ···	C3	227 ···	E3	1024 ···	400
196 ···	C4	228 ···	E4	1280 ···	500
197 ···	C5	229 ···	E5	1536 ···	600
198 ···	C6	230 ···	E6	1792 ···	700
199 ···	C7	231 ···	E7	2048 ···	800
200 ···	C8	232 ···	E8	2304 ···	900
201 ···	C9	233 ···	E9	2560 ···	A00
202 ···	CA	234 ···	EA	2816 ···	B00
203 ···	CB	235 ···	EB	3072 ···	C00
204 ···	CC	236 ···	EC	3328 ···	D00
205 ···	CD	237 ···	ED	3584 ···	E00
206 ···	CE	238 ···	EE	3840 ···	F00
207 ···	CF	239 ···	EF	4096 ···	1000
208 ···	D0	240 ···	F0	4352 ···	1100
209 ···	D1	241 ···	F1	4608 ···	1200
210 ···	D2	242 ···	F2	4864 ···	1300
211 ···	D3	243 ···	F3	5120 ···	1400
212 ···	D4	244 ···	F4	5376 ···	1500
213 ···	D5	245 ···	F5	5632 ···	1600
214 ···	D6	246 ···	F6	5888 ···	1700
215 ···	D7	247 ···	F7	6144 ···	1800
216 ···	D8	248 ···	F8	6400 ···	1900
217 ···	D9	249 ···	F9	6656 ···	1A00
218 ···	DA	250 ···	FA	6912 ···	1B00
219 ···	DB	251 ···	FB	7168 ···	1C00
220 ···	DC	252 ···	FC	7424 ···	1D00
221 ···	DD	253 ···	FD	7680 ···	1E00
222 ···	DE	254 ···	FE	7936 ···	1F00
223 ···	DF	255 ···	FF	8192 ···	2000

付録 6　SYMDEB[DEBUG]コマンド抜粋

コマンド	英　語	機　能
A	Assemble	アセンブル
D	Dump	ダンプ(メモリ内容表示)
G	Go	実行
Q	Quit	終了
R	Register	レジスタ表示
U	Unassemble	アンアセンブル

```
                                   起動
A>SYMDEB ⏎ ──────────────────────┘
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [8086]
-A ⏎ ─────────────────────────────────────────── アセンブル
394E:0100 MOV AX,0A000
394E:0103
-
-D100,105 ⏎ ────────────────────────── ダンプ
394E:0100  B8 00 A0 00 00 00                                8. ...
-G103 ⏎ ───────────────────────────── 実行
AX=A000  BX=0000  CX=0000  DX=0000  SP=C5A1  BP=0000  SI=0000  DI=0000
DS=394E  ES=394E  SS=394E  CS=394E  IP=0103   NV UP EI PL NZ NA PO NC
394E:0103 0000          ADD     [BX+SI],AL                         DS:0000=CD
-U100,103 ⏎ ───────────────────────────────── アンアセンブル
394E:0100 B800A0        MOV     AX,A000
394E:0103 0000          ADD     [BX+SI],AL
-R ⏎ ───────────────────────────────────────── レジスタ表示
AX=A000  BX=0000  CX=0000  DX=0000  SP=C5A1  BP=0000  SI=0000  DI=0000
DS=394E  ES=394E  SS=394E  CS=394E  IP=0103   NV UP EI PL NZ NA PO NC
394E:0103 0000          ADD     [BX+SI],AL                         DS:0000=CD
-Q ⏎ ──────────────────┐
                        │
A>                      │
                       終了
```

日本語索引

英語索引

著者略歴

藤田英時(ふじた・えいじ)

株式会社アクセス・グループ代表取締役。福岡市生まれ、現在、パソコン関連の書籍・雑誌記事の執筆・編集、マニュアルのテクニカルライティングなどで活躍中。著書・訳書は次のとおりハード・ソフトの各種入門書からアセンブリ言語プログラミング法まで幅広くあり、その解説のわかりやすさには定評がある。

やさしいホログラフィー入門(編訳)システムソフト、BASICホームコンピュータプログラム集(訳)システムソフト、PU-ハンドブック(編著)システムソフト、BASICビジネスプログラム集(訳)システムソフト、PC-Tech Know 9800(共著)システムソフト、マシン語クックブックI(共著)システムソフト、ユーティリティ・ハンドブック98(共著)システムソフト、オフィシャル・プリンタ・ハンドブック(編著)エー・アイ・ソフト、マシン語マジックブックI(共著)日本ソフトバンク、英語でパソコン・ウォッチ(編著)エー・アイ・ソフト、マシン語マジックブックII(共著)日本ソフトバンク、クリエイティブ・コンピュータ・グラフィックス(訳)リットーミュージック、パソコン・マニュアルの読み方(著)日本実業出版社、PC-98LTガイドブック(著)アスキー出版局、MS-DOSユーティリティを作る(著)ラジオ技術社、楽しく学ぶ1—2—3(共著)エーアイ出版、98知りたいことがわかる本(著)エーアイ出版、Lotus 1-2-3関数・マクロ活用事例集(共著)エーアイ出版、最新MS-DOS入門百科(著)旺文社、最新MS-DOS応用百科(著)旺文社

いちばんわかりやすいMS-DOSマクロアセンブラ入門

著　者　**藤田英時**　ⒸEiji Fujita 1987

発行者　**田村正隆**

発行所　**株式会社ナツメ社**
東京都千代田区神田神保町1-52(〒101)
電話　03(291)1257(代表)
振替　東京3-58661

印　刷　**ラン印刷社**

製　本　**三修紙工**

ISBN4-8163-0755-9 Printed in Japan.

いちばんわかりやすい
MS-DOS
マクロアセンブラ入門
実用プログラムを開発しながら
マシン語の知識が身につく本

発行――1989年7月20日
著者――藤田英時
発行者――田村正隆
発行所――株式会社ナツメ社
郵便番号101
東京都千代田区神田神保町1-52
電話〈03〉291-1257
振替 東京3-58661
〈落丁・乱丁本はお取り替えします〉

定価――**1860円**

[目次より]



いちばんわかりやすい MS-DOS マクロアセンブラ入門

実用プログラムを開発しながらマシン語の知識が身につく本

藤田英時＝著

ナツメ社

ISBN4-8163-0755-9 C2054 P1860E

ナツメ社 | 定価1860円（本体1806円）